Czech Technical University in Prague
Faculty of Nuclear Sciences and Physical Engineering

DOCTORAL THESIS

Distributed Data Processing in High Energy Physics

Prague 2018 Dzmitry Makatun

Bibliografický záznam

Autor	Mgr. Dzmitry Makatun, České vysoké učení technické v Praze, Fakulta jaderná a fyzikálně inženýrská, Katedra matematiky
Název	Distribuované zpracování dat ve fyzice vysokých energií
Druh práce	Disertační práce
Studijní program	Aplikace přírodních věd
Studijní obor	Matematické inženýrství
Školitel	doc. Michal Šumbera, CSc., DSc., Ústav jaderné fyziky, Akademie věd České republiky
Školitel specialista	Jérôme Lauret, Ph.D., RHIC/STAR Experiment, Physics Department, Brookhaven National Laboratory, USA
Konzultant	doc. Hana Rudová, Ph.D., Fakulta informatiky, Masarykova univerzita, ČR
Akademický rok	2017/2018
Počet stran	196
Klíčová slova	Distribuovaný výpočet, grid, vyvažování zátěže, plánování procesů, optimalizace, toky v sítích, big data

Bibliographic Entry

Author	Mgr. Dzmitry Makatun, Czech Technical University in Prague, Faculty of Nuclear Sciences and Physical Engineering, Department of Mathematics
Title	Distributed Data Processing in High Energy Physics
Degree Program	Application of natural sciences
Field of Study	Mathematical engineering
Supervisor	Michal Šumbera, Ph.D, Nuclear Physics Institute, The Czech Academy of Sciences
Supervisor specialist	Jérôme Lauret, Ph.D, RHIC/STAR Experiment, Physics Department, Brookhaven National Laboratory, USA
Consultant	Hana Rudová, Ph.D, Faculty of Informatics, Masaryk University, Czech Republic
Academic Year	2017/2018
Number of Pages	196
Keywords	Distributed computing, large scale computing, grid, data intensive applications, load balancing, job scheduling, planning, network flow, data production, big data

Acknowlegement

This thesis would not have come to life without the excellent guidance of my three supervisors through the challenging journey of my studies.

I am sincerely thankful to Michal Šumbera for the opportunity to enter the fascinating world of computational High Energy Physics and for the strong support of his valuable advice and wisdom since the very beginning of the study.

I am tremendously grateful to Jérôme Lauret, who sparked the research project and helped to keep it focused on real-life tasks. His inspirational leadership, broad expertise and kind attitude, spiced with a refreshing portion of humor, gave momentum to my research and helped me to develop the confidence to attack problems which may seemed unsolvable at first.

I would like to express my deepest gratitude to Hana Rudová, who played a key role in my study. Her professionalism, attention to important details and patience, all combined with her warmhearted approach, were crucial to keep my research moving on track.

I would also like to thank Jana Bielčíková, who has had multiple roles during these years in our Ultra-relativistic Heavy Ion Group, for her always helpful advice and caring management. I am thankful to the current and former members of our scientific group at Bulovka, especially to Michal Zerola, Filip Křížek, Pavol Federič and Dagmar Adamová for the friendly atmosphere and thoughtful discussions.

A special mention goes to the members and students of SITOLA laboratory and Department of Computer Systems and Communications at Faculty of Informatics of Masaryk University in Brno. My work trips there had a huge impact on my study thanks to the productive meetings and the creative team spirit.

I would like to acknowledge the Institute of Physics of the Czech Academy of Sciences for providing access to their computing facility for running simulations.

Last, but not least, the priceless support and encouragement of my dear family, especially my mother Maryna Makatun and my grandmother Nadezhda Ariko, has helped me to keep moving forward in my pursuit towards science.

Declaration of Originality

This doctoral thesis contains results of my research carried out at the Nuclear Physics Institute of the Czech Academy of Science between years 2011 and 2018. Hereby I declare that this thesis is my original authorial work. All sources, references, and literature used or excerpted during the preparation of this work are properly cited and listed in the bibliography. I further state that no part of this thesis or any substantially the same has been submitted for any qualification other than the degree of Doctor of Philosophy at the Czech Technical University in Prague.

Abstract

In the era of big data, the scale of computations and the amount of allocated resources continues to grow rapidly. Large organizations operate computing facilities consisting of tens of thousands of machines and process petabytes of data. A lot of effort was made recently to optimize the design of such computer clusters, resource management and corresponding computing models including data access and job scheduling. Scientific computing (e.g. High Energy and Nuclear Physics (HENP), astrophysics, geophysics, genome studies) appears at the forefront of big data advancement. Due to the scale of computations, these fields rely on aggregated resources of many computational facilities distributed over the globe. Those facilities are owned by different institutions and include grid, cloud and other opportunistic resources. Orchestration of massive computations in such a heterogeneous and dynamic infrastructure remains challenging and provides many opportunities for optimization.

One of the essential types of the computations in HENP is distributed data production where petabytes of raw files from a single source have to be processed once (per production campaign) using thousands of CPUs at distant locations and the output has to be transferred back to that source. Similar workflows can be found in other distributed data-intensive applications. The data distribution over a large system does not necessarily match the distribution of storage, network and CPU capacity. Therefore, bottlenecks may appear and lead to increased latency and degraded performance.

The problems of job scheduling, network stream scheduling and data placement are interdependent, but combined into a single optimization problem become computationally intractable in a general case. In practice, there are multiple middleware components with different (overlapping) scopes of the system, each providing optimization for its sub-problem. The examples include workload management systems, job schedulers, data transfer services, data management systems, etc. The end-to-end optimization becomes a matter of an interplay between middleware components. Automated high-level orchestration can improve such interplay and reduce the effort for system tuning.

The main goal of this thesis is to explore and develop a new approach to optimization of large-scale data-intensive computations in HENP in gen-

eral, and the STAR experiment in particular. As the result of this thesis, we propose a new high-level orchestration approach for distributed data production. The underlying mathematical model introduces a new application of network flow maximization algorithms. In our approach, a central planner defines how much input and output data should be transferred over each network link in order to maximize the computational throughput. Such plans are created periodically for a fixed planning time interval using up-to-date information on network, storage and CPU resources. The complexity of each planning cycle depends on the number of sites and network links but not the number of jobs. This allows to manage extensive datasets for processing in large-scale infrastructures efficiently. The centrally created plans are executed in a distributed manner by dedicated services running at participating sites.

A wide scope of simulations based on the log records from real systems and monitoring were performed for this Ph.D. thesis. The simulations have shown that the proposed approach systematically provides a significant improvement in makespan and processing throughput compared to other simulated traditional techniques.

Abstrakt

Zpracování rozsáhlých dat vyžaduje čím dál více výpočetních prostředků. Velké moderní organizace provozují počítačová centra obsahující desítky tisíc strojů a zpracovávají petabajty dat. Proto je v současnosti v popředí zájmu optimalizace návrhu výpočetních clusterů, způsobu ovládání zdrojů, přístupu k datům a plánování úloh. Mezi hlavní zájemce o zpracování rozsáhlých dat patří mimojiné i vědecké experimenty ve fyzice vysokých energií (High Energy and Nuclear Physics, HENP), astrofyzice, geofyzice nebo v genetice. Vzhledem k objemu vstupních dat a potřebných výpočtů se často projevuje tendence kombinovat zdroje z mnoha vzdálených výpočetních clusterů distribuovaných po celém světě. Ty často patří různým organizacím a jsou tvořeny systémem propojených a spolupracujících počítačů uspořádaných do různých architektur (grid, cloud). Koordinace distribuovaných výpočtů v takovémto heterogenním a dynamickém prostředí je tak velkou výzvou s velkým prostorem pro optimalizaci.

Jedním z hlavních typů výpočtů v HENP je distribuovaná produkce dat, kdy musí být petabajty dat naměřených detektorem jednorázově zpracovány (během tzv. produkční doby) ve vzdálených centrech. Výsledná data jsou buď poslána zpět do původního centra nebo jsou uložena na jiném místě. Podobné postupy lze nalézt i v jiných oblastech, kde se pracuje s rozsáhlými soubory dat. Distribuce dat v rozsáhlých systémech nemusí nutně odpovídat distribuci kapacity datových úložišť, sítě a procesorů. Vzhledem k tomu může dojít ke vzniku kritických míst s nízkou průchodností, což vede k poklesu efektivity.

Otázky plánování úloh, přenosu dat a jejich umístění spolu vzájemně souvisí. Při spojení do jednoho problému se tak ale v obecném případě stávají výpočetně neřešitelnými. Existuje nicméně mnoho druhů tzv. middleware, které poskytují optimalizaci pro jistý dílčí problém. Sem patří vyvažovače zátěže, plánovače úloh, přenos dat, správa dat atd. Komplexní optimalizace se tak stává záležitostí interakce mezi jednotlivými složkami middlewaru. Automatická koordinace může tuto interakci zlepšit, a tak i snížit nutnost lidského zásahu do systému.

Hlavním cílem předložené práce je prozkoumat a vyvinout nový přístup k optimalizaci distribuovaných výpočtů s rozsáhlými daty v HENP obecně a zejména se zaměřením na experiment STAR. Výsledkem výzkumu prove-

deného v rámci této disertace je návrh nového přístupu ke koordinaci distribuované produkce dat založeného na použití algoritmů pro řešení problému maximalizace toku v sítích. Centrální plánovač určuje kolik vstupních a výstupních dat musí být přenášeno přes každou síťovou linku tak, aby byla maximalizována propustnost výpočtů. Plánování se pravidelně realizuje pro pevně daný časový úsek s přihlédnutím k aktuálním informacím o stavu sítě, datových úložišť a procesorů. Důležité je, že složitost každého výpočetního cyklu závisí na počtu výpočetních zdrojů a síťových linek a není přitom závislá na počtu výpočetních úloh. Tento systém umožňuje efektivně zpracovat velké množství dat ve velkých infrastrukturách. Centrálně vytvořené plány se pak za pomoci speciálních služeb pracujících ve zúčastněných výpočetních centrech vykonávají distribuovaně.

Pro tuto studii byla v rámci této disertační práce provedena široká škála počítačových simulací založených na skutečných záznamech pocházejících ze žurnálů (logů) monitorovacích systémů. Simulace ukázaly, že navrhovaný přístup poskytuje systematické a výrazné zlepšení doby zpracování dat, a tudíž i produktivity výpočetních clusterů ve srovnání s jinými tradičními metodami.

Contents

List of Figures

List of Tables

Glossary

ALICE A Large Ion Collider Experiment. A HENP experiment at LHC. 84

ATLAS A Toroidal LHC ApparatuS. A HENP experiment at LHC. 83

big data a paradigm describing a new generation of technologies and architectures, designed to economically extract *value* from very large *volumes* of a wide *variety* of data, by enabling high-*velocity* capture, discovery, and/or analysis. 38

BNL Brookhaven National Laboratory. 83

cache policy heuristic used to select an entry to evict with regard to cache cleanup. 152

central storage the main data storage facility of a HENP experiment. Typically, is situated close to the detector (at Tier-0 site) and permanently stores all the data related to the experiment. 105

CERN European Organization for Nuclear Research (derived from French: Conseil Européen pour la Recherche Nucléaire). 83

cloud a model for enabling access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction. 48

cluster group of closely linked computers, working together through fast local area networks; in opposite to Grid, resources are not geographically spread. 47

CMS Compact Muon Solenoid. A HENP experiment at LHC. 83

CP constraint programming. 89

CSP constraint satisfaction problem. Consists of variables, their domains and constraints in form of mathematical expressions over variables. 89

DAG Directed Acyclic Graph. 44

data center a cluster dedicated to storage, processing and providing access to large amounts of data. 48

data prestaging placing input data close (in the access sense) to the point of computation before it starts. Also referred as prefetching. 73

data production an organized processing of raw data (from a detector) in order to produce data ready for analysis (reconstructed physical events). Also referred as reconstruction or preprocessing. 84

DDM Distributed Data Management (system). 73

DPS Data Placement Service. 73

graph data structure that holds a collection of vertices and a collection of edges that connect pairs of vertices. 44

grid distributed and dynamic computing environment consisting of various loosely coupled resources acting together to perform large tasks. 48

HENP High Energy and Nuclear Physics. 83

HPC High Performance Computing. 41

IT Information Technology. 40

job (or computational job) an atomic unit of computational work for scheduling. The terminology varies in related publications (also depends on specifics of computing platform). We use the term from the perspective of scheduling in grid computing. 55

lfn logical file name. 121

LHC Large Hadron Collider. A particle accelerator facility at CERN. 83

LHCb Large Hadron Collider beauty. A HENP experiment at LHC. 84

load balancing methodology to distribute workload across multiple resources to achieve optimal utilization. 64

makespan total time spent on execution of a set of tasks. 128

min-cost max-flow minimum cost maximum flow. A problem of finding a maximum flow with a smallest possible cost over a given network. 117

MPI Message Passing Interface. Message passing standard on a wide variety of parallel computing architectures. 45

NP-hard non-deterministic polynomial-time hard. Class of problems from computational complexity theory that are, informally, "at least as hard as the hardest problems in NP". 54

pfn physical file name. 121

planning selection and organization of actions in order to reach the goal or change of the system. 51

queue structure which stores tasks waiting for execution. Tasks are selected according to the applied dispatching rules. 64

resource entity which executes, processes or supports the task (e.g. CPU for a computational job, network link for data transfer, storage for data placement). 56

RHIC Relativistic Heavy Ion Collider. A particle accelerator facility at BNL. 83

scheduling allocation of resources to planned tasks over given time periods. 51

site a separate computing facility in grid (e.g. cluster, server, supercomputer, data center, external cloud). 105

STAR Solenoidal Tracker at RHIC. A HENP experiment at RHIC. 83

task an atomic unit of schedulable work. We use the term in a more general meaning, which does not necessarily refer to a computational job executed at a machine, but includes other types of atomic work, e.g. a transfer of a file over a network link. 55

Tier-0 site of the highest level in a hierarchical grid architecture. In HENP, the main computing facility of an experiment, typically close to the detector. 86

Tier-1 sites of the second highest level in a hierarchical grid architecture. In HENP, regional data centers which disseminate / aggreagte data to / from computing facilities of lower tiers and permanently store significant replicas of experimental data. 86

Tier-2 level of sites in a hierarchical grid architecture. In HENP, large computing facilities of scientific institutes and universities. 87

Tier-3 level of sites in a hierarchical grid architecture. In HENP, such sites have no formal roles assigned, which provides flexibility to join and leave for specific tasks. 87

unary resource resource with an ability to execute only one task at any time. Sometimes is also called a serial or a disjunctive resource. 91

weighted graph graph with an associated label (weight) to every edge. It is often used in networking where weight represents bandwidth of a link. 90

WLCG Worldwide LHC Computing Grid. 85

WMS Workload Management System. 71

workflow set of tasks (including computational jobs), which may have specific dependencies between each other. The terminology varies in related publications (also depends on specifics of computing platform). We use the term from the perspective of scheduling in grid computing. 55

List of Notations

α_i	processing time per unit of data at site c_i
β	average output-to-input size ratio of data production jobs
b_l	bandwidth of network link l
C_i^{out}	total size of output files of currently running jobs at site c_i
C_{max}	makespan
$c_i \in C$	set of computational sites
cap_e	capacity of edge e
$cost_e$	cost assigned to edge e
ΔT	planning time interval
$Dest_j$	set of sites where the output file of job j has to be delivered (to a single site in the set)
$Disk_i$	size of local disk at site c_i
δ	upper limit for storage usage
$d_i \in D$	set of dummy edges from the computing sites to the sink in the input planning problem
$\overline{d}_i \in \overline{D}$	set of dummy edges from the source to computing sites in the output planning problem
ε	target precision for initial data distribution planning
$e \in E$	set of edges in the network of the min-cost max-flow problem (includes dummy edges)
Φ	total network flow (from source to sink)
F_l	counter of the remaining input data to be sent over link l during the execution of the current plan
$\overline{F_l}$	counter of the remaining output data to be sent over link l during the execution of the current plan
f	file id
f_j^{in}	input file of job j
f_j^{out}	output file of job j
$flow_e^{in}$	solution of the input planning problem, which assigns amount of input data to be transferred over each edge e
$flow_e^{out}$	solution of the output planning problem, which assigns amount of output data to be transferred over each edge e
H	cache hits

H_d	cache data hits (cache hits per megabyte of data)
I_i^{in}	total size of input files prestaged in a local queue at site c_i
I_i^{out}	amount of output data (of previously finished jobs) which is staged to be transferred from site c_i
j	computational job
K_i	maximum amount of input data that can be initially placed at source site c_i
k_i	amount of input data staged at source site c_i
$\overline{k}_i$	amount of output data which can be accommodated at destination site c_i during current planning cycle
L_i^{from}	set of outgoing links of site c_i
L_i^{to}	set of incoming links of site c_i
$l \in L$	set of real network links
N_{cache}	number of files transferred from cache
N_{req}	total number of requests
N_{set}	number of unique filenames
$NCPU_i$	number of CPUs provided for data production at site c_i
o_l	time required to transfer a unit of data over link l
p_j	duration of job j
p_{fi}^F	duration of placement of file f at site c_i
$q_i \in Q$	set of dummy edges from the source to source sites in the input planning problem
$\overline{q}_i \in \overline{Q}$	set of dummy edges from destination sites to sink in the output planning problem
R_i	free storage space at site c_i
S_{fi}^F	start time of file f placement at storage of site c_i
S_j^J	start time of job j
S_{fl}^T	start time of file f transfer over link l
S_{cache}	amount of data transferred from cache in bytes
S_{req}	total amount of transferred data
S_{set}	total size of unique files
$Sources_j$	set of sites where the input file of job j is initially placed
$size_f$	size of file f
$size_{avg}^{in}$	average size of input files
$size_j^{in}$	size of the input file of job j

$size_j^{out}$	size of the output file of job j
T_{est}	estimated makespan for a given assignment of tasks to resources
T_i^{CPU}	estimated completion time of site c_i
T_l^{link}	estimated completion time of link l
w_i	demand for input data at cite c_i
$\overline{w}_i$	estimated amount of output data to be transferred from site c_i
X_{fl}	boolean variable indicating if file f is transfered over link l
Y_{ji}	boolean variable indicating if job j is executed at site c_i
Z	total amount of data to be processed in a data production campaign

1 Introduction

1.1 Motivation

Modern experiments in High Energy and Nuclear Physics (HENP) engage processing of large volumes of data derived from complex detectors and simulations [4, 5, 6]. Data-intensive distributed computing has become an essential part of this scientific field. The computational infrastructure of the largest HENP experiments running at BNL RHIC and CERN LHC, spans across the globe and features many tens of facilities processing petabytes of data annually [7, 8]. Similarly, many other scientific fields, such as astrophysics, biology and Earth science to name a few, as well as industries and commercial companies handle enormous volumes of data.

When running data-intensive applications on distributed computational resources long I/O overheads may be observed as access to remotely stored data is performed. Uncoordinated concurrent data access over a shared network can lead to an increased latency [9, 10, 11]. Latency and bandwidth can become the major limiting factors for the overall computation performance and can reduce the CPU time / wall time ratio due to excessive I/O waits [12]. In such case the benefit of usage of distributed resources is hindered due to network congestion [13]. In particular, a small fraction of computational jobs which cannot access data efficiently ("stragglers") can increase an overall makespan dramatically. Intuitively, those jobs could be completed faster if scheduled to different resources (even if they wait in a queue) or if the data are efficiently prefetched beforehand. For this reason, optimization of data access and management is an important issue when defining a computing model of a HENP experiment [4, 14], setting up a new computational facility [15] or upgrading/tuning an existing one [16, 17].

Data processing can be divided into several phases with its own specifics of jobs. This thesis focuses on a particular phase which is called data preprocessing in big data terminology [18]. In this phase a large set of input data undergoes a single pass of processing and produces output data which is further utilized in other phases. The particular example considered in this study is data production (see Section 5) in HENP. However, the devised techniques can be further extended to other data processing workflows. In data production, raw data from a detector are processed in order to reconstruct physical

events which are then analyzed by scientists. The data are stored in the main data center close to the detector and are further distributed for remote processing. The processing has a data level of parallelism, which means that it can be divided into independent computational jobs applying the same processing on different subsets of data. The data production is performed by campaigns, when a recently accumulated dataset has to be processed on an available set of resources. Such campaign typically lasts several months and processes hundreds of terabytes of data. Sometimes, the processing is repeated (after a significant time) when it allows improving the quality of the output. The reconstructed data can be effectively utilized only after an entire campaign is finished. For this reason, it is highly desirable to execute data production with the shortest possible makespan at given resources. Also importantly, data production (and simulations) makes the largest computing demand of the STAR and other HENP experiments [19]. Its optimization may lead to a huge overall saving in computational power.

However, common computing models (see Section 2.2) and optimization approaches (see Section 4) to data-intensive applications do not match the data production case well. This is mainly due to the specific properties of such workflow: (a) the data originates from a single source (detector) and has to be disseminated to geographically distributed resources, (b) there is no data re-usage across data production jobs within a single campaign. Despite the named differences from other types of computations (analysis, simulations), data production is often approached by general scheduling techniques, and that may lead to sub-optimal performance and increase the requirements for computational infrastructure (CPU, network, storage).

Data aware schedulers (see Section 4.1) typically exploit (a) spatial and/or (b) temporal data locality for optimization. In the first case (a), the jobs are allocated where the data are already present or as close as possible (e.g. using an estimation of prestaging overhead). In the second case (b), the jobs sharing the data are grouped together in order to reduce the number of required transfers. If we offload data production from a central facility to remote resources, which do not have the data already prestaged, the data locality cannot be exploited. This is because the jobs process non-overlapping portions of data stored at the same location. Therefore, a permutation of data production jobs between sites would not help.

Common job schedulers (see Sections 3.4 and 4.1) do not consider such details as data transfer routing, bandwidth sharing and storage scheduling. Optimization of data transfer and prestaging for scheduled jobs is delegated to other components of the system (see Sections 4.2 and 4.3). As the result, concurrency between newly scheduled jobs may lead to system overload if network or storage bottlenecks are present. However, these components cannot influence the job scheduler decisions. Moreover, optimization of data transfer applied in practice is often limited to parameter tuning (e.g. block size, number of active streams, etc.), replica selection and compliance with predefined deadlines.

Being an important aspect of data access optimization, virtually every data replication strategy (Section 4.4) aims to improve availability of popular data with respect to (upcoming) requests . However, this principle cannot be directly applied to data production, since the raw data are processed exactly once within a single campaign, and the campaigns are separated by large time gaps making the re-use of cached data irrelevant. The raw data experience infrequent access compared to other data types, e.g. reconstructed data. Therefore, it would be impractical to keep many replicas of raw data in the system.

In practice, static replication approaches are applied to raw data: a fixed number of backup copies is stored. For example, LHC experiments follow a standard strategy whereas each raw file is persistently stored at the central (Tier-0) site and has two replicas at distinct regional centers (Tier-1) [4]. However, a significant fraction of computational power is dispersed at smaller national and institutional facilities (Tier-2,3) or external clouds. Therefore, the data locality can be exploited for data production only if the computation is limited to Tier-0,1 resources. The offloading of data production to other resources would allow to decrease its makespan and speed up the delivery of reconstructed data for user analysis. Such offloading requires data prestaging (in and out) at remote sites. Similarly, the STAR experiment at BNL [20] stores the raw data at its central facility and offloads data production to remote sites upon agreement.

To enable efficient data production at remote sites, experiments often use custom setups for each given distributed infrastructure [19, 21]. When there are few remote sites involved in the data processing, the load can be

tuned manually and simple heuristic may work, but, as the number of sites grows and the environment is constantly changing (site outage, fluctuations of network throughput and CPU availability), an automated planning of workflows becomes a necessity. An important example arises from a workflow optimization which was done for the inclusion of the ANL computational facility into the data production of the STAR experiment [22]. In this case, the throughput of a direct on-demand network connection between BNL and ANL was not sufficient to saturate all the available CPUs at the remote site. At the same time, the lack of available storage space at ANL did not allow to prestage the data in advance. An optimization was achieved by feeding CPUs at ANL from two sources: directly from BNL and through LBNL as an intermediate site. Such counter-intuitive solution was established after the complex analysis of the workflow performance in the multi site system with respect to storage and network. This example illustrates an efficient use of indirect data transfers which cannot be derived using simple heuristics.

To summarize, the management of distributed data intensive computations has been an important research topic for decades and its relevancy still grows as the big data paradigm spreads its fields of application. Due to its complexity, the problem of end-to-end optimization is decomposed into several sub-problems. Optimization is often provided by separated components at distinct levels. The global optimization can be achieved by ensuring interplay and coherence between components, parameter tuning and a high level orchestration. Also, there exist case specific solutions which combine several sub-problems in order to achieve better optimality. To our best knowledge, no such solutions are adjusted to specifics of distributed data production. Moreover, as it was discussed above and illustrated in the ANL case, the optimization approaches designed for common workflows with spatial and/or temporal data locality do not fit the data production case well.

1.2 Contribution

The main goal of this thesis is to explore and develop a new approach to optimization of large-scale data-intensive computations in HENP in general, and STAR experiment in particular. In this thesis we propose a novel high-level orchestration approach for distributed data production. The approach exploits specific properties of the workflow in order to consider CPU, network

and data scheduling within a single tractable optimization problem. Data distribution is dynamically adjusted during computation in order to make the best use of provided resources. The underlying mathematical model introduces a new application of network flow maximization algorithms to job scheduling and load balancing problems. Rethinking of existing job scheduling policies which shifts the priority towards efficient data management has shown its potential for large-scale data-intensive computations such as data production in HENP. With the help of simulations based on data from real systems our approach is validated and compared to scheduling techniques used in practice. Also, the realistic simulations study the influence of network performance on the overall computational efficiency.

In earlier work, which was completed in collaboration between BNL and NPI CAS, a new approach for optimization of data transfer in distributed systems was proposed by Michal Zerola [11]. Our initial idea was to extend that previous work, and include CPU and storage scheduling into consideration. We have proposed our first scheduling approach based on constraint programming in [23]. That work allowed us to study the problem of joint co-scheduling of jobs, transfers and data placement and revealed the potential for optimization of resource usage. However, the limitations of such approach were encountered. The underlying constraint satisfaction problem in its general formulation appeared excessively complex. After refining the problem formulation, we have discovered that it can be efficiently solved with network flow maximization algorithms. We have presented the initial ideas of the new model at the MISTA 2015 conference [24]. Shortly after, we have completed the model with solving procedures, implementation, execution algorithm and performed simulations of base use cases [25]. This article also summarizes the most important contributions of the thesis as a detailed journal publication which was recently accepted after significant revisions along the time. Further, we continued the development of the planning approach based on network flow maximization, extending it to more use cases and evaluating in large-scale simulations. In [26] we studied the influence of background network traffic and simulated data production in Tier-1 network of one of the largest HENP experiments. We added load balancing between multiple data sources and optimization of initial data distribution in work [27]. There we also performed simulations of data production in randomly generated large-

scale grids imitating real infrastructures. In [28] we extended our approach to deal with data replication across the system and simulated a heterogeneous grid of Tier-0,1,2,3 sites.

In order to further optimize data access for computations, we have studied applicability of known caching algorithms to data access patterns in HENP [29].

The list of the corresponding publications with the primary authorship of the author of this thesis is also provided in Appendix A.

1.3 Structure of the Thesis

In Chapters 2 – 5 we discuss the state of the art of distributed data-intensive computing. First, in Chapter 2 we consider big data paradigm, its applications and technologies in order to provide a broader context for our work. Then, Chapter 3 focuses on modern approaches to resource management and job scheduling in distributed systems. Chapter 4 is dedicated to optimization of data access for computations. The specifics of computing in HENP is summarized in Chapter 5.

Then, in Chapter 6, we apply constraint programming to build an initial model which allows to study the potential for optimization of data production. The lessons learned from the study allowed us to better understand the problematics and discover the limitations of the first model. Based on the first experience, we have developed a novel job scheduling approach.

The main contribution of the thesis is presented in the two subsequent chapters. In Chapter 7 we propose a novel approach to data production planning based on network flow maximization. We formalize the details of a considered problem and present the base model, solving method and plan execution. Then we gradually extend our approach to reason on more aspects of data production management. In Chapter 8 the new approach is validated in simulations based on data obtained from real HENP computing systems. There, our planner is compared against scheduling policies currently used in practice. The extensive simulations consider various use cases. We start with base use cases needed to understand the behavior of the system, proceed with realistic infrastructures and confirm the results over a wide set of randomly generated large-scale setups. In the simulations we also consider background network traffic, balancing between multiple data sources, initial

data distribution and data replication.

In Chapter 9 we provide an additional study of caching algorithms for distributed data processing in HENP, which helps to further improve the efficiency of data access.

A conclusion to the work and future outlook are given in Chapter 10.

A list of related publications by the author of this thesis is provided in Appendix A.

2 Big data paradigm

The combined amount of data accumulated in digital world over past decades, as well as rapidly increasing speed of its generation, has overwhelmed the capacity of traditional data processing/management approaches. The ability of pioneering enterprises (both scientific and industry) to access/manage/process data at a new unprecedented scale allowed to extract new, previously not available, value out of extremely rich, detailed and diverse datasets. Early success examples have ignited the emergence of big data paradigm. The quantitative growth of data available to organizations has lead to a qualitative shift of its combined value: scientific (e.g. statistical) methods of data analysis have been adopted by other communities, which allowed them to gain valuable insights into their domains and make better informed and timed decisions. Technologies dealing with big data have grown into a vast ecosystem and provided new functionality to enterprises and individuals: modern society has a commodity to access and utilize (sometimes implicitly) the information at a scale and speed unimaginable few decades ago.

While still remaining a hot research topic due to a huge variety of particular applications, the term *big data* captures specific properties of modern data-intensive applications. Multiple concurrent definitions exist which focus on different aspects of the phenomena. The first definition of the trend (without spelling "big data" itself) was given as early as 2001 using the concept of 3 Vs: *Volume*, *Velocity* and *Variety* [30]. Later the concept was extended to 4 Vs adding *Value* as another important aspect, making the most commonly accepted definition of the big data term. For instance, in 2011 International Data Corporation (IDC) [31] has defined big data in the following way:

> "Big data technologies describe a new generation of technologies and architectures, designed to economically extract *value* from very large *volumes* of a wide *variety* of data, by enabling high-*velocity* capture, discovery, and/or analysis."

Similarly, the National institute of Standards and Technology (NIST) defines big data as [32]:

> "Big data is where the data volume, acquisition velocity, or data representation limits the ability to perform effective analysis using

traditional relational approaches or requires the use of significant horizontal scaling for efficient processing."

The aforementioned concept of 4 Vs can be summarized as follows:

Volume. There is no fixed margin which separates "normal data" from big data. The datasets referred in related studies vary from terabytes and petabytes to exabytes, while the sheer volume of data aggregated on the Internet and faced by modern search engines counts in zettabytes. Big data is when the size of the data itself becomes part of the problem [33].

Velocity. The speed at which the data are generated e.g. by scientific experiments, individual users, commercial transactions, mobile devices and sensors introduces additional challenges to its capture, storage and processing.

Variety. The data coming from different sources can be structured, semi-structured or unstructured (e.g. text messages or multimedia). Extraction of value in such case often requires additional processing including filtering, format transformation, consistency checks, redundancy removal and error correction.

Value (or Veracity). The reliability, credibility and representativeness of data may vary. For example, a large fraction of social media content, web and e-mails is made up by spam; clickstream and mobile traffic are subject to noise. However, information of interest can still be extracted from a huge volume of data with a low value density. A complex workflow and data itself are prone to errors of various nature, therefore, obtaining reliable outcomes of analysis is not a trivial task.

The data-value chain in big data applications consists of the following stages: generation, acquisition, storage, analysis [18].

In this chapter, we provide a short overview of big data as a concept, its application domains and technologies. More detailed overview including the history of the field, discussion on definition, classification with many particular examples can be found in [18, 34, 35, 36, 37]. The paper [38] provides an insight into current agenda and perspective research topics as of 2016.

2.1 Applications

Information Technology (IT) solutions addressing the new data-intensive paradigm were pioneered by scientific communities such as High Energy and Nuclear Physics (HENP), astronomy and biology [39], as well as innovative companies such as Google, Amazon, Twitter, Facebook, Microsoft, Oracle, Apache, IBM and many others. Those solutions, while intensively evolving during past years, are being widely adopted and further developed by more organizations and companies.

In a collection of articles [40] authors combine experience from multiple domains (Earth, environment and health care studies) in order to demonstrate how these scientific fields were transformed by the exponential increase in scientific data. They also demonstrate how the big data related technology has influenced the scientific and scholarly communication, emphasizing the trend of governments and funding agencies investing into open access to both scientific data and publications.

Big data technologies have found its application in various scientific domains. Let us quickly go over the map of modern big data science. In computational biology repositories containing petabytes of data about genes, proteins, small molecules and medical records are shared by many research groups around the world. Data analysis allows to study genes, tumors, live organisms, viruses, protein interaction, brain activity, etc. [39]. In medical science and health care sector large volumes of data are shared for collaboration on clinical trials, personnel training, epidemics detection and monitoring, development of new diagnostics/treatment/drugs/vaccines [41]. Tens of petabytes of data belonging to climate science mostly origin from satellite instruments and numerical climate model simulations, but also include other diverse instrumental data. Two major challenges in this domain were highlighted in [42]. The first is to ensure that the expanding volumes of data are easily and freely available to enable new scientific research. The second is to make these data and the results useful to a broad interdisciplinary audience, because there is a growing interest by other communities of researchers. Social sciences are also among the most benefited from the rapid growth of available data. While it operated surveys of thousands or so in the past, now the researchers can harvest millions of social media post, huge quantities of social networking information, location, search queries, related datasets from

biological sciences and much more [43]. In astrophysics big data technologies are applied to process the data originating from telescopes, satellites, gravitational wave detectors [44] and large-scale simulations. The Berkeley Open Infrastructure for Network Computing (BOINC) developed for Search for Extraterrestrial Intelligence (SETI) [45] was a milestone in the development of volunteering computing. NASA and ESA space agencies utilize big data approaches to process information delivered by active missions as well as to design new missions [46]. Big data paradigm became an essential part of modern HENP experiments, it is discussed in more details in Section 5.

In industry, big data technologies have found a broad spectrum of applications. The examples include search/indexing systems [47], online retail [35], social networks [48, 49], multimedia services [50], recommendation engines [51], business intelligence [50], customer analytics [52], finance [35], logistics [53] and engineering [35].

2.2 Technologies

2.2.1 Big data and HPC

It is important to notice, that two "worlds" of big data technologies can be distinguished based on the type of utilized software stack. The first one addresses big data problems using (customized) software which is generally attributed to High Performance Computing (HPC). The second one uses the technologies initially designed for big data problems specifically. Such division can be explained through history and application specifics. By the time when the need for data-intensive computing has emerged, many corresponding scientific fields already had well developed software/infrastructure for large-scale distributed computing (HPC). Therefore, it was a natural choice to gradually extend existing frameworks to the new data-intensive problems. This also allowed to keep interoperability of the system for both data and CPU intensive applications. At the early stage, when the requirements and applicability of data-intensive computing where not yet clearly understood, there was no necessity to re-design already mature solutions of HPC. Alternatively, other enterprises (such as Google and Yahoo), designed their own solutions to address data-intensive computing on commodity hardware specifically. As the concepts of big data became better understood and

wide spread, such solutions were adopted and further extended by more organizations. A good side by side comparison of HPC and "pure" big data approaches to data-intensive computing can be found in [54] and [55]. Both of the approaches have many similarities and adopt concepts from each other, therefore, a convergence is envisioned by many researchers in the field. As of today, according to the studies, the HPC like approach provides better performance optimization for specific applications, while the "pure" big data ecosystem provides greater flexibility and fault tolerance.

The driving force behind the development of big data technologies is the need to develop scalable solutions for parallel data processing. Technologies for data-intensive computing can be classified into three categories: file systems, programming models and databases. Some of the solutions are highly specialized for particular types of applications, while the others are designed for greater generality. Often, the file systems and computing models are co-designed to achieve optimization for a particular set of applications.

2.2.2 File systems

Storage solutions addressing the challenges of big data scale became a cornerstone of the technology of the new paradigm. According to CAP theorem [56], a storage system can ensure only two out of three desired properties: Consistency, Availability and Partition tolerance. For instance, traditional relational databases operating relatively small datasets provide consistency and availability (while typically running on a single server). ACID (Atomic, Consistent, Isolated, Durable) [57] systems were designed to ensure consistency and partition tolerance but feature a limited availability (eventual availability) which can hinder throughput of dependent applications. Such systems are useful in cases with a moderate load but strong requirements on data consistency (e.g. financial transactions). The next step into big data era was the emergence of BASE (Basic Availability, Soft-state, Eventual consistency) systems [58]. Such systems sacrifice strong consistency in order to ensure availability and partition tolerance. This approach matches the demand for high I/O flow in distributed data-intensive applications and became common in past years.

Google File System (GoogleFS) [59] and MapReduce [47] introduced by

Google were among early big data technologies which shaped the concept of future relevant solutions. Both GoogleFS and MapReduce were developed in parallel and are mutually optimized. GoogleFS was designed as a scalable distributed file system for data-intensive applications hosted on unreliable commodity hardware. It implies little data reuse within a single application run. In GoogleFS multiple data servers storing the data are managed by a master server which holds metadata. The master redirects user requests to data servers, controls locks, manages namespace, guides data replication, balances load and performs garbage collection. In this way the GoogleFS separates file system control, which passes through the master, from data transfer, which passes directly between data servers and clients. It treats component failures as a norm rather than an exception, optimizes for huge files that are mostly (concurrently) appended to and then (sequentially) read, and provides fault tolerance by constant monitoring, data replication and automatic recovery. MapReduce is a programming model and an associated implementation for processing and generating large datasets. A User specifies a *map* function that process key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Programs written in such style are automatically parallelized and executed on a large cluster of (commodity) machines. The runtime system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. It was proven that MapReduce can emulate any distributed computation [60].

Hadoop [61] is an open source implementation of MapReduce. It was co-designed with the Hadoop Distributed File System (HDFS) [62] which is an open source version of GoogleFS. Since it is an open-source project it has been widely adopted in big data community and further derivative versions with different focuses have forked.

A broad spectrum of distributed file systems exists, however, none of them can be seen as a one-size-fits-all solution, as they are optimized for specific use cases. For example, Network File System (NFS) [63] has a simple architecture where a single server exports a local directory tree to a number of clients. The capability of the single server limits the scalability of such approach. Andrew File System (AFS) [64] distributes the responsibility for

file system subtrees to different servers. However, static partitioning of a file system tree limits its applicability. XrootD [65] uses tree based routing in hierarchy of servers and is optimized for high-throughput access to HENP datasets. The CernVM File System [66] is designed to distribute software binaries. Lustre [67] is optimized as a scratch space for cooperating applications on supercomputers. Ceph [68] file system has distributed meta-data architecture which overcomes the bottleneck of the central meta-data server (as in GoogleFS, HDFS) at the cost of increased complexity of the system. This file system has been recently used in STAR experiment [69, 70]. Decentralized file systems (GlusterFS [71]) allow clients to compute the location of data/meta-data by means of a distributed hash-table. Many of the distributed file systems (e.g. Ceph) provide build-in data redundancy where the same data are replicated across multiple servers in order improve data availability and sustain frequent joins and leaves of the nodes. However, in such cases, the data placement is defined by the internal logics of the file system and does not necessarily match data access patterns or availability of computing power. A compact overview of most popular distributed file systems can be found in [72] and [73].

Large organizations, such as experiment collaborations in HENP, store data in a global federation of various cluster file systems rather than in a single, globally distributed file system [72]. Data Federation seeks to integrate data management and data access, resulting in a global management system that can handle replication and transfer of data to storage or to running applications [74]. All LHC experiments currently rely on XrootD for the wide area delivery of the data files, though several use the local experiment catalog for discovery. Such heterogeneity of file systems and limitations of coordination should be accounted for when optimizing data-intensive computations across multiple distinct facilities.

2.2.3 Computing models

We have already discussed MapReduce and Hadoop (in conjunction with their relevant file systems GoogleFS and HDFS), let us briefly summarize several other common computing models.

Dryad [75] is a general purpose computing engine developed by Microsoft. It considers a computation as a Directed Acyclic Graph (DAG): programs

are graph vertexes and communication channels are graph edges. The graph may change during the execution.

HPC infrastructures/software were traditionally designed for scientific applications aiming towards high-end computing capabilities. Computing and data elements are typically separated in HPC. The workload consists of multicore jobs using MPI [76], OpenMP [77] or PGAS [78] APIs for communication. The jobs are typically allocated and managed by batch scheduling systems such as PBS/Torque [79] (gang scheduling). Generally, these systems focus on managing computing slots. Various approaches are applied to enable and optimize data-intensive jobs on HPC infrastructures. For instance, pilot jobs generalize the concept of a placeholder to provide multi-level and application level scheduling. This enables data-aware scheduling and data staging. Modern grid middleware tools (e.g. Condor [80], DIRAC [81]) provide data access and management support for distributed MPI applications. More details on conjunction of big data and HPC paradigms can be found in [54].

Spark [82, 83] generalized MapReduce and multiple specialized computing models. While previous solutions provided fault tolerance through data replication, Spark has a different approach. It introduced Resilient Distributed Datasets (RDD) [84] — a read only collection of objects partitioned across a set of machines that can be rebuilt if a partition is lost. Users can explicitly store an RDD (in memory or at persistent storage) across machines and reuse it in multiple parallel operations (similar to MapReduce tasks). RDDs achieve fault tolerance through a notion of linage: if a partition of an RDD is lost, the RDD has information about how it was derived from other RDDs and is able to rebuild just that partition. By default, RDDs are lazy and ephemeral. That is, partitions are materialized on demand when they are used in a parallel operation. RDDs are the best suited for batch applications that apply the same operation to all elements of a dataset, but less efficient for applications that make asynchronous updates to shared state (such as storage systems for web applications or an incremental web crawler). Due to its efficient (in-memory) data sharing, linage-based fault recovery, generality (applicable to batch, iterative and streaming computing) and open-source development model, Spark has seen a wide usage in big data community recently.

While the computing models described above are mostly focused on batch processing, there is also a number of solutions for real-time (streaming) data processing. Such solutions are used for processing messages and updating databases, continuous querying on data streams, parallelizing an intense query on the fly (e.g. a search query) and more [85]. Kafka [49, 86] and Storm [87] are example of big data engines for stream processing.

Database functionality is often implemented on top of the two above (e.g. BigTable [88] on GoogleFS and MapReduce, HBase [89] on Hadoop). There are also independent database solutions for big data as MongoDB [90] and Cassandra [91].

Many of the computing models implement their own application-level resource management. This allows to dynamically schedule jobs to available resources. Hierarchical scheduling is applied to enable concurrency control between multiple frameworks which are sharing resources. In such system, application-level schedulers communicate with a central component (master scheduler) which orchestrates allocation. The common examples of such scheduling include YARN [92] and Mesos [93]. A summary of popular big data computing models and corresponding resource management can be found in [85]. We further discuss the topic of resource management in more details in Sections 3 and 4.

3 Job scheduling in distributed computing

The programming model defines the data processing workflow and its division into parallel jobs. Many existing big data frameworks implement their own application-level scheduling. The *elastic* frameworks (Hadoop, Dryad) can scale its resources up and down, i.e., it can start using nodes as soon as it acquires them and release them as soon as the job is completed. In contrast, a *rigid* framework, such as MPI, can start running its jobs only after it has acquired a fixed quantity of resources, and cannot scale up dynamically to take advantage of new resources or scale down without a large impact on performance. A resource management system enables execution of jobs from multiple frameworks on shared resources. There are multiple functions which a resource management should provide. It includes resource discovery, scheduling, allocation, monitoring, fault recovery and compliance with authentication and security mechanisms. In this thesis we focus on job scheduling and data access aspects of resource management.

3.1 Distributed platforms

While big data frameworks vary in their architecture, they commonly utilize parallel processing where a large computation is split into many parallel jobs. Those jobs are executed on various types of distributed computational resources. Such resources are organized in multiple different ways depending on their purpose, hardware, ownership and legacy aspects. In this regard, such terms as supercomputers, computer clusters, data centers, grid and cloud are often referred. Let us briefly explain this terms here.

A *supercomputer*, according to the traditional notion, has many processors connected by a local high-speed computer bus. At present time, the term mostly refers to specialized systems designed to address specific computationally intensive problems.

In *cluster computing*, a large number of processors are used in proximity to each other. In other words, computer cluster consists of a set of loosely or tightly connected computers that work together so that they can be viewed as a single system. The components of a cluster are usually connected to each other through fast local area networks, with each node (machine) running its own instance of an operating system. In most circumstances, all of

the nodes use the same hardware and the same operating system. Dedicated middleware (such as RMS and DMS standing for Resource/Data Management Systems) allows to use the machines aggregated into a cluster for the common goal.

A *data center* can be viewed as a computer cluster dedicated to storage, management, processing and providing access to large volumes of data.

Grid is a collection of computer resources from multiple locations to reach a common goal. In the grid computing approach, the processing power of many computers, organized as distributed, diverse administrative domains, is opportunistically used whenever a computer is available. The grid can be thought of as a distributed system with non-interactive workloads that involve a large number of files. Therefore, grid computers tend to be more heterogeneous and geographically dispersed (thus not physically coupled) than cluster computers. The resources that are integrated into grid are typically dedicated computational platforms, either high-end supercomputers or general-purpose clusters. Examples include LHC Computing Grid [4], TeraGrid [94], Open Science Grid [95], European Grid Infrastructure (EGI) [96], NorduGrid [97] and MetaCentrum in Czech Republic [98].

Grids can be classified by topology into hierarchical, P2P (federated), hybrid and arbitrary graphs. *Hierarchical topology* assumes a structured network, generally in the form of a tree or a star. It is adopted in many scientific projects and a multi-tier data grids (such as initial computing model of LCG). In *P2P* architectures nodes act autonomously, possessing functionality of both servers and clients at the same time. P2P offers more flexibility in communication among components compared to hierarchy, however, it adds complexity to the design and management. *Hybrid architectures* combine at least two other architectures. For example, it can be a hierarchical topology combined with a P2P-like communication of components at the same level of hierarchy. Finally, in *arbitrary graph* topology nodes are freely connected and can wary in roles. A scale-free network of servers and clients connected via the Internet can be an example. Another example is a grid which was initially designed as hierarchical but more elements and connections were added arbitrarily over time.

Cloud is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks,

servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. The cloud infrastructure can be viewed as containing both a physical layer and an abstraction layer. The physical layer consists of the hardware resources that are necessary to support the cloud services being provided, and typically includes server, storage and network components. The abstraction layer consists of the software deployed across the physical layer, which manifests the essential cloud characteristics [99].

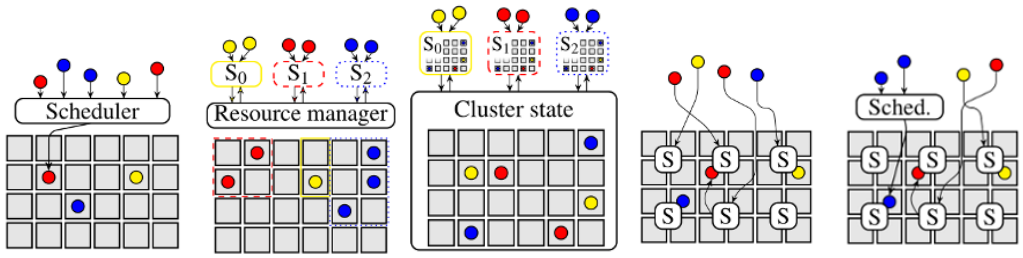
The arising concept of *cloud federations* involves usage of resources provided by multiple clouds from distinct provides in order to execute large-scale computations. This concept resembles the main aspects of grid computing by the definition.

Cloud computing is based on several other computing research areas such as High Performance Computing (HPC), virtualization, utility computing and grid computing [100]. Viewed in a broader sense, the concepts of grid and cloud computing have many similar features. Like the grid, the cloud is a utility computing model that involves a dynamically growing and shrinking collection of heterogeneous, loosely coupled nodes, all of which are aggregated together and present themselves to a client as a single pool of compute and/or storage resources. Most of the major differences come from the difference in respective clients. In grid, clients run massive workflows, generally non-interactive (e.g. large-scale scientific experiments), and the resources are owned and administrated by multiple organizations. In cloud, the resources owned by a single enterprise are tenanted by thousands (or millions) of clients running smaller workflows, but often sensitive to response time. Nevertheless, in both grids and clouds there is a common need to manage large facilities; to define methods by which consumers discover, request, and use resources provided by central facilities; and to implement highly parallel computations that execute on those resources [101].

According to many surveys, such as [102], from the scheduling perspective, there are more common features among the discussed platforms then specific ones. Moreover, it is increasingly common to consider hybrid infrastructures, in which in-house resources are complemented with resources from cloud or grid platforms.

3.2 Scheduler architectures

Figure 1 illustrates various types of job scheduling architectures [1]. Gray



(a) Monolithic (b) Two-level (c) Shared-state (d) Distributed (e) Hybrid

Figure 1: Architectures of job schedulers [1].

boxes represent resources, circles represent jobs and S_i denotes schedulers. The architectures can be divided into five groups:

Centralized (Monolithic). A single scheduler makes placement decisions for all jobs and resources.

Two-level (Hierarchical). The scheduling over the entire system is achieved through communication in a hierarchy of schedulers (typically two levels of hierarchy). Two cases are important. First, in big data paradigm, multiple computing engines (e.g. Hadoop, Spark) have their own application-level schedulers integrated. Those schedulers are orchestrated by a central component (see Figure 1b) which resolves conflicts and controls fair sharing of resources. The second case is hierarchical scheduling in grid. There, each site (cluster) can have a local scheduler (e.g. Condor [80]) which communicates with a central scheduler (e.g. Condor-G [103]). The jobs are submitted either directly to local schedulers, or to the central scheduler which redirects the jobs to the local ones. In this case the resources are statically partitioned between local schedulers, and the central scheduler enables inter-site job submission. In both cases, the central scheduler plays an active role – allocates jobs based on its policy.

Shared-state (Decentralized). Multiple schedulers are plugged into the central component. But this time the central component enables

operation of schedulers but does not make placement decisions on its own. For example, it can be a distributed database storing the state of available resources. Multiple schedulers act concurrently, each having a smaller scope of the problem: a specific type of jobs or a subset of resources. The examples are Omega [104], Apollo [105] and Nomad [106].

Distributed (Fully-distributed). There is no central component in a fully-distributed system. Each computing node has its own scheduler, which can schedule incoming jobs at the node or forward it to another one. The absence of a single bottleneck is an advantage. The downside is that each scheduler has a limited knowledge of the jobs/resources in the system, therefore the global optimality can be compromised. Sparrow [107] is an example of such distributed scheduler.

Hybrid. The load is split between a centralized and distributed components.The central component can utilize sophisticated algorithms for scheduling of critical jobs, but use simple distributed heuristics for low priority jobs. Hybrid scheduling is implemented in Tarcil [108], Mercury [109] and Hawk [110] schedulers.

3.2.1 Examples

There is a variety of batch scheduling systems designed for MPI workloads at a centrally managed cluster. Such systems feature centralized architecture of a scheduler. The most common examples are: LoadLeveler [111], LSF [112], Maui [113], NQE [114], PBS [115], Torque [79], Condor [80] and Moab [116]. Most of them were designed in 90-s and are still in use at modern scientific computing centers, their development continues to stay up-to-date with requirements of today's applications.

A grid scheduler can be seen as a higher level on top of local schedulers. The examples of grid resource managers are Condor-G [103], Globus [117] and Legion [118]. The core effort of Condor-G's planning and scheduling was dedicated to matchmaking, i.e., finding appropriate resources and setting the corresponding environment, security and data access for computational jobs at distributed systems. The matchmaking is performed based on job requirements being analyzed towards properties and policies of resources (participating clusters). Condor-G also provides functionality for job

migration across computing clusters, error recovery, backups in progress of computations, and scheduling of complex workflows described by DAGs.

Google's Borg [119] is an example of a centralized cluster management system. Each job runs in one cell (cluster). A cell consists of a set of machines, a centralized controller called Borgmaster, and an agent process called Borglet that runs on each of the machines. The Borgmaster consists of two processes: the main process and a separate scheduler. The main process handles client RPC (Remote Procedure Calls) and manages the state of all of the objects in the system (machines, jobs). The scheduler asynchronously scans the jobs in the queue, checks their feasibility to find machines on which the job could run and scores the machines to find the best placement for a job. The scoring takes into account user-specified preferences but is mostly driven by the built-in scheduling criteria. The Borglet starts/stops/restarts jobs, manages local resources by manipulating OS kernel settings, manages debug logs, and reports the state of the machine to the Borgmaster. The Borgmaster process is replicated several times, each replica maintains an in-memory copy of the system state. In case of failure of the active master, one of the replicas is selected using election mechanism to replace it fast. While the Borg is a proprietary software, its concepts and exploitation experience were utilized in the development of the next generation of cluster management systems Kubernetes [120] and Firmament [121] which became open-source.

Mesos [93] is an example of a two-level cluster management system. Authors define it as a platform for sharing commodity clusters between multiple diverse cluster computing frameworks, such as Hadoop and MPI. It introduces a two-level scheduling mechanism called resource offers. There is a master process that manages slave daemons running on each cluster node, and frameworks that run jobs on these nodes. Each framework registers its own scheduler with the Mesos master. The master, knowing the state of all nodes, decides how many resources offer to each framework (e.g. to achieve fair share). The framework's scheduler decides which resources to accept and which jobs to run on them (e.g. to achieve a better data locality).

Apache YARN [92] (Yet Another Resource Negotiator) has a similar architecture consisting of a centralized Resource Manager (RM), multiple Application Managers (one per each application/framework) and a Node Man-

ager (NM) running at each node. Both AMs and NMs communicate to RM through heartbeat messages in order to reduce communication load. The main difference is that YARN is a request-based (in contrast to offer-based Mesos) resource manager: RM allocates resources to AMs upon request specifying amount/properties of the demanded resources. This makes the system more centralized (in some sources categorized as monolithic), since a single component makes job placement decisions. Also, while Mesos has a fixed pool of framework schedulers, YARN allocates AMs dynamically – one for each running job.

Omega [104] is an example of a shared-state scheduler with the main focus on scalability. Multiple independent schedulers are granted access to the entire cluster and compete for resources in a free-for-all manner using a shared-state. A resilient master copy of the resource allocations is maintained using optimistic concurrency control. Once a scheduler makes a placement decision, it updates the shared copy of the cluster state in an atomic commit. At most one such commit will succeed in the case of conflict. The schedulers can choose to use incremental transaction (which will accept all but the conflicting changes) or all-or-nothing transactions (either all jobs are scheduled together or none are). In case of refused transactions, the scheduler does another iteration on remaining jobs and updated cluster state. Therefore, the two-level scheme's centralized component is simplified to a persistent data store with a validation code that enforces common rules. While the architecture is highly scalable, a heavy load may lead to a large number of conflicts which would force scheduler to re-run often.

3.3 Scheduling models

Given the scale of computations and amount of utilized resources in big data applications, even a small improvement in efficiency of resource utilization can have a large value. Increasing data processing efficiency allows enterprises to save investments in hardware and electric power, as well as accomplish computations in shorter time.

Most resource management systems have a modular design where separate components take care of particular functions: communication, setting runtime environment, monitoring, scheduling and etc. This allows to implement various logic in the scheduling component without changing the rest

of the system. However, the design of the resource management system influences the design of the scheduling algorithm as it defines what inputs are available and what outputs are expected.

A general formulation of the resource allocation problem can be found in [12, 122, 123]. Expected Time to Compute (ETC) [123] and Total Total Processor Cycle Consumption (TPCC) [123] models are often used to formulate job scheduling problems (especially in more early works). Such models can be seen idealistic as they assume exact knowledge of the performance of each job on each resource in advance. Moreover, important details of various modern systems are not included into classical ETC and TPCC formulations (multicore jobs; usage of various types of resources, e.g. CPU, memory and network; monetary cost; fairness; data locality). Later practical works tend to utilize customized models and consider interaction with monitoring services which provide up-to-date information on the changing state of jobs and resources. However, since the job scheduling problem is NP-hard there is no general formulation / solution which would remain computationally acceptable and match all modern large-scale distributed systems. Each particular research addresses a certain combination of resource platform (single machine, cluster, grid, cloud, dedicated supercomputer or their combination), workflow (type of jobs) and solution (allowed operations, solving method and optimization criteria). Given the variety of such combinations, parallel job scheduling in distributed systems has been a fertile research ground for some decades. Thus, the number of papers with solutions and surveys related to the area is enormous. Those factors led to segregation into many research groups with focus on particular problem instances. Such situation significantly complicates the navigation in the existing knowledge base as well as collaboration between the researchers. Moreover, the terminology and taxonomies often vary between research communities. As a result, it is difficult to trace and generalize (mathematical) similarities between considered problems and proposed solutions. Such problem is explicitly discussed in [102] where the authors propose a universal taxonomy for parallel job scheduling in distributed systems, based on previously proposed taxonomies and an extensive analysis of publications in the field. For the mentioned reasons we will not focus on any particular formal model in this section, but rather discuss important elements and goals for development of such

a model. More different taxonomies and surveys targeting specific categories of platforms (yet sometimes overlapping) exist, as for example: grid resource allocation [122], data grids [12, 124], workflow management systems in grid [125], scientific workflows in IaaS clouds [126], general resource scheduling in clouds [127, 128]. It is also worth to mention, that according to [129], despite a wide selection of advanced scheduling algorithms, many smaller infrastructures, which do not explore optimization, rely on well established but yet simplistic approaches.

3.3.1 Terminology

Since terminology varies in publications from a wide range of related areas, here we define the terms used in this thesis. In particular, the terms "job" and "task" are ambiguous and their precise definition depends on the domain (e.g, scheduling or computing) and the considered platform (cloud, grid, cluster, operating system, etc.). For the purpose of scheduling, it is important to distinguish which term corresponds to the atomic unit of work to be considered.

As our research studies distributed data production in HENP, where grid computing terminology is more common, we use the term *(computational) job* to refer an atomic unit of computational work for scheduling. An important implied example is a computing program which is executed on a single computational node, occupies a fixed amount of resources, processes/produces a definite portion of data and can be scheduled separately from its peers (independently of other jobs). A computational *workflow* consists of multiple jobs, which may have specific dependencies between each other. We use the term *task* in a more general meaning, which does not necessarily refer to a computational job executed at a machine, but includes other types of atomic work, e.g. a transfer of a file over a network link.

It is important to mention, that many studies on scheduling in big data and parallel computing use the term "task" to define the smallest unit of schedulable work (similar to our usage of "job"). In such terminology, a "job" consists of multiple tasks and is similar to our usage of the term "workflow". For example, a MapReduce [47] job consists of multiple "map" and "reduce" tasks. Also, a multicore job is often considered as a group of tasks each executed on a separate core. While different terms reflect important specifics

of platforms, the scheduling principles often remain similar regardless of terminology. In this thesis, we use the terminology defined in the previous paragraph for consistency, but provide additional notes, where necessary, in order to avoid confusion.

Next, let us define less ambiguous terms used in this thesis. *Resource* is an entity which executes, processes or supports the task (e.g. CPU for a computational job, network link for data transfer, storage for data placement). *Submission time* is the time when the scheduler has received the job to be scheduled (e.g. from the user). *Start time* is the time when the job starts its execution in the resource. *Waiting time* is the time interval between submission and start of the job. *Completion time* is the time when the job has completed its execution and releases the resource to other jobs. *Duration* of a job is defined as the time interval between its start and completion times. Strictly spiking, it can only be measured after the job is completed. However, an estimation is implied when referring a duration of a not yet completed job (e.g. a job being scheduled). In literature, duration is also often referred as *execution time* or *processing time*. *Slowdown* is the ratio of the time spent by the job in the system (from its submission to completion, including waiting time) to its duration.

The choice of a scheduling approach greatly depends on the properties of expected workload and target platform. There is always a trade-off between the generality of anticipated workloads/platforms, algorithm complexity and efficiency for the primary case. Next in this section, we discuss properties of jobs, resources and desired optimizations which are important to define a scheduling model. Let us start with the properties of computational jobs.

3.3.2 Jobs

Scheduling granularity. There are distinct types of workflows with respect to how their jobs are allocated. The first type, multicore (MPI) workflows[1], requires intensive communication and synchronization among jobs during the execution. Therefore, all the jobs of a workflow should be allocated simultaneously (gang-scheduling). If there are not enough resources to

[1]see Section 3.3.1 for discussion of terminology

schedule such workflow immediately, the scheduler has to wait for more resources to become free. These may imply putting a lock on already acquired resources and preventing other workflows from using them (e.g. [130]). The second type is more flexible, but dependencies between jobs still exist which can be specified with a Directed Acyclic Graph (DAG). The workflow may consist of sequential, parallel and synchronization stages, which implies ordering of some of the jobs and data exchange (e.g. Dryad [75]). The third type of workflows is not constrained by communication or dependencies. Any number of jobs can be executed at a time. The scheduler can start dispatching jobs of a workflow as soon as any number of appropriate resources become available. For example, the map jobs of the MapReduce workflow do not require to communicate with each other. The only requirement is that the reduce jobs should be able to access the map output, therefore they could be scheduled simultaneously or after the map jobs. Another example is so-called "bag-of-tasks" workloads, where jobs have no dependencies among each other and can be executed out of submission order. Such model provides more flexibility in scheduling and allows for better packing of jobs on resources over time.

Job flexibility. A computational job may require a strictly defined amount of resources for its execution (rigid) or can change its requirements according to scheduler decision at startup (moldable) or during the runtime (malleable). In case of evolving jobs, an application-level scheduler decides how much resources should be acquired by the job.

Hierarchy of jobs. The jobs within a mixed workload may vary in their priority. For example, user-facing services, interactive jobs or crucial production services are more important than batch data processing jobs. Schedulers implementing jobs hierarchy aim to provide shorter queuing time, better placement choices or fault tolerance to jobs with higher priority.

Timing constraints. In some cases users, may want to specify deadlines for their jobs. The deadlines can be implemented either as hard or soft constraints. Real time and interactive jobs are the most sensitive to timing which requires the scheduler to support performance guarantees (Service

Level Agreement aware schedulers). The jobs with no crucial time requirements are referred as best-effort jobs.

3.3.3 Resources

Let us discuss representation of resources in scheduling models.

Heterogeneity. All the resources in the considered problem may be equivalent (e.g. a uniform computer cluster, or cores of the same CPU), or resources may vary in their properties (e.g. grid). In the latter case, the jobs may explicitly specify the requirements on resources (e.g. platform, CPU frequency, memory size, operating system, installed software, access permissions). It is a duty of a scheduler to check the consistency of such requirements and identify a set of resources matching each job.

Slot-based vs. elastic resource representation. The pool of resources can be divided into slots (e.g. a fixed number of cores and amount of memory) which can place a single job. If the job does not consume exactly the amount of resources in the allocated slot the resource trashing occurs. Alternatively, resources can be dynamically shared between multiple jobs allocated to the same resource (e.g. virtual machine). According to many studies, elastic scheduling leads to a better resource utilization [85]. However, concurrency of jobs allocated to the same resource may lead to interference and performance degradation. Therefore, an isolation of concurrent processes is an important aspect of resource management. In order to mitigate the interference, it is preferable to mix jobs of distinct types (CPU-intensive, memory-intensive and network-intensive) at a machine.

Over submission. A scheduling system may have multiple job queues for different subsets of the machines. In such case, the same job can be inserted into multiple queues. The job waits until it is scheduled for the first time and then it is removed from the rest of the queues. Such approach is used to implement certain policy features and improve job placement, however, it sophisticates the operation.

Resource leasing. In Infrastructure as a Service (IaaS) cloud computing, the user defines how many resources (e.g. virtual machines) to lease. Depending on the pricing model, the price for resource lease may also vary in time. Also, multiple types of VMs can be available for deployment with different price and combination of parameters (e.g. CPU speed, memory size). When scheduling a workload on a cloud from behalf of a user, it becomes a part of a problem to define how many (and what type of) VMs should be requested at a given time, depending on the workload properties.

Speculative execution. If a workflow consists of multiple parallel jobs it has to wait until every one of them is completed. Under a heavy load or constrained resources, some of the jobs can get sub-optimal placement. The progress of a job can be delayed by slower machines, data access overhead, interference with other jobs, resource congestion or failures. Such jobs are often called *stragglers*. Therefore, critical jobs can be executed in several concurrent instances at different machines. Also, when the most of the jobs in a workflow are completed and the stragglers become easy to detect, some scheduling systems (as Spark [83] and several implementations of MapReduce [47])[2] can resubmit the unfinished jobs for speculative execution. The output is taken from whichever instance of the job completes first and the remaining instance is terminated. Such approach allows to improve the response times but leads to excessive computational work.

Preemption. A scheduler may consider not only the new jobs from the queue but also reconsider already scheduled and running jobs. It may become advantageous to terminate some of the active jobs to provide a better allocation to high-priority jobs. It is often used as a mechanism to enforce fairness and better resource matching. Preempted jobs are restarted later when more resources become available, or, in some systems, they may be migrated to other resources. However, killing jobs in the middle of their progress leads to a waste of computational work. To mitigate the losses, active jobs are often assigned costs of preemption proportional to CPU time already spent, i.e. their priority increases with time (hysteresis).

[2]The corresponding publications use different definitions of jobs and tasks. See discussion in Section 3.3.1.

3.3.4 Optimization

Advanced scheduling algorithms consider multiple solutions to the scheduling problem and select the best one according to the scheduling goal. It is typically done with the help of an objective function which scores a solution based on its properties. Similar functions can be utilized as metrics for comparison of different algorithms against each other or for monitoring purposes. Let us discuss the most commonly used optimization criteria. Data access, being an important topic for this thesis, is extensively discussed in Section 4, while the rest of the criteria are summarized below.

Makespan is the time interval between start and completion time for a set of jobs. According to survey [102] 60 % of 100 most cited research papers on job scheduling consider makespan minimization (sometimes along with other metrics).

Flowtime is the sum of completion times of all the jobs (see [131]). The smaller flowtime indicates faster completion of some jobs. It can be achieved by scheduling shorter jobs ahead of longer ones. Flowtime is often used as a secondary metrics to quantify the Quality of Service (QoS) in earlier works. More recent works tend to utilize custom metrics for this purpose, or explicitly incorporate compliance with the Service Level Agreement (SLA) into their approaches.

Total weighted completion time is similar to the flowtime, but the summed completion times of jobs are weighted according to their priority (see [131]). Minimization of this metrics promotes faster completion of more important jobs.

Response time of a job is the time interval between its submission and completion. Average weighted response time is calculated as a weighted sum of response times of all the jobs divided by the sum of the weights (see [131]). This metric is applied in scheduling of interactive applications.

Waiting time of a job is the time interval between its submission and the execution start. The average weighted waiting time is defined as a weighted

sum of waiting times of all the jobs divided by the sum of the weights. Aside from the quality of a schedule, in real systems, this metric allows to study latencies introduced by the scheduling process itself.

Resource utilization can be defined as a ratio of effectively used resources (e.g. CPUs processing data) to the total resource capacity (e.g. total number of CPUs, including ones currently idle). Such metric is important for heavily loaded systems where it is highly desirable to utilize resources to full capacity. Of cause, the capacity should not be exceeded to avoid performance degradation.

Matching proximity compares a current job allocation to a "perfect" allocation where each job is executed at a resource providing the shortest execution time. Matching proximity is computed as a ratio of sums of job execution times under the current and the "perfect" allocations (see [131] on page 614). This metric is rarely used in recent works.

Fairness. When resources are shared between multiple users (virtual organizations/frameworks/jobs) it is important to maintain fair usage. One of the common examples is that a large job should not monopolize an entire cluster, delaying the progress of other (small) jobs. There is a vast spectrum of approaches addressing this issue. A user can be limited in the number of resources it is allowed to utilize simultaneously. Or the share of allocated resources can be proportional to the fraction of user's tasks over the total number of active tasks. It can also be normalized with respect to a hierarchy of priorities between users. More sophisticated algorithms consider the history of previous allocations and multiple types of resources (CPU time, memory, network) [85, 132]. Users may be allocated "budgets" which are spent when using resources. One of the common approaches is to specify an objective function for optimization which expresses fairness. The policy towards fairness can be soft (not dispatch additional tasks from the user who currently exceeds its fair-share) or hard (preempt active tasks to maintain fairness). A policy may allow to execute tasks above calculated quota when there are free resources (no concurrency), but at risk of those being preempted later by new incoming tasks from different users.

Monetary cost. An infrastructure may consist of resources with different ownership and pricing models, including external commercial clouds. Those resources can have distinct costs of execution of user's jobs. A scheduler can consider placement of jobs with respect to the cost of computation. Typically, the goal is to either minimize the overall cost of scheduled computations or to find the best trade-off between processing time and expenses. For example, a user can specify a budget constraint for the submitted computations. It is important to distinguish scheduling approaches which consider an actual monetary cost of computations and those which use the term "cost" in general meaning: in order to consider several optimization goals within a single objective function.

Energy efficiency. Since electricity bills make a significant fraction of operational costs of clusters and data centers the research on energy consumption reduction is trending. This topic is addressed by research on both energy efficient hardware and scheduling. When a computational cluster is not utilized to the full capacity it is advantageous to aggregate active jobs on a subset of machines and power-off the idle ones as well as related equipment (e.g. network switches). It is important to maintain a stable level of operation and not be miss-directed by short-term load fluctuations, because excessive shutdowns/start-ups increase energy consumption and reduce lifetime of the equipment. It is also important to notice, that concentration of jobs over fewer machines increases the probability of a correlated failure. An alternative approach is Dynamic Voltage and Frequency Scaling (DVFS) which allows different power modes in hardware. With DVFS the energy consumption and performance are adjusted according to the resource load.

Network traffic. One can measure the amount of data transferred over the network during processing. Smaller traffic indicates better job placement decisions with respect to data locality, which allows to reduce network load and decrease I/O waiting time. This metrics is typically used as a secondary criterion.

Correlated failures. In large-scale clusters of commodity hardware, failures are rather a norm than an emergency. In opportunistic computing,

resources can join and leave at any moment of time. For this reason, critical services are often duplicated. For instance, a critical job can run in multiple instances, one of which is selected to be active (master) and the rest are backup. The redundancy can also be used to distribute the load between service instances and reduce response time (e.g. the backup instances serve read-only requests). In order to improve fault-tolerance, such instances are allocated to minimize the probability of correlated failures. To achieve this, the jobs are placed on distinct physical machines, located at distinct racks, powered by independent sources, connected to different network routers or, even to distinct geographical locations. A similar approach is often applied to data replicas in distributed storage systems [68].

In addition, custom functions may quantify other goals of optimization, e.g. violation of deadlines, number of migrated or preempted jobs, job priority compliance. Multicriteria optimization can consider a combination of several metrics. Also, in order to compare different scheduling algorithms the time required to make a scheduling decision is measured.

3.4 Scheduling methods

The problem of job scheduling in large-scale distributed systems is challenging. The search space of such problems is huge as it includes all possible assignments of jobs to resources and the order of jobs. It also features complex constraints and optimization criteria. Some of the common optimization criteria are in mutual conflict, e.g., makespan and fairness, or energy efficiency and tolerance against correlated failures. Moreover, the parameters of resources and jobs are subject to changes and wrong estimations. As the result, it is not just computationally problematic to find an optimal solution, but sometimes even impossible to strictly define it in practical cases. For this reason, heuristics and meta-heuristics are widely applied in practice. Such methods allow to find efficient solution within a reasonable time without the necessity to perform complete search. In this section, we summarize the most widely applied solving methods to job scheduling problems in distributed systems, which include: immediate mode heuristics, batch mode heuristics, backfilling, local search, population-based meta-heuristics and several other. Detailed surveys and taxonomies of solving methods applied to job scheduling problem can be found in [12, 102, 123, 129]. More compact overviews of

the of the most popular methods are provided in [133, 134].

3.4.1 Immediate mode heuristics

Immediate mode heuristics [135] schedule one job at a time. The job waits in a queue of unscheduled jobs until it is processed by the scheduler. The job priorities can be implemented as queue ordering. Some schedulers operate on multiple queues arranged by types of jobs or subsets of resources. The queue-based approach is well suited for parallel processing by multiple schedulers in hierarchical architectures. However, the schedulers do early placement decisions which restrict choices for further waiting jobs. This category includes:

- *Opportunistic Load Balancing (OLB).* Assigns each job to the earliest idle resource. When multiple resources are available at the same time OLB selects an arbitrary one. Such method provides uniform load balancing, however, does not consider any additional optimization.

- *Minimum Completion Time (MCT).* The job is assigned to a resource which provides the earliest completion time. MCT considers the time when the resource completes previously assigned jobs (ready time) and the execution of the job at the resource. However, MCT does not aim to provide the best fit with respect to execution time of each job.

- *Minimum Execution Time (MET).* The job is assigned to the resource with the smallest execution time for this job. MET does not take the availability of resources into account. Therefore, neither load balance nor makespan are optimized.

- *Switching Algorithm (SA).* Tries to combine strong sides from MCT (load balancing) and MET (minimal execution times). It divides the minimum ready time by the maximum ready time among the resources in order to estimate current load balance. If the ratio is low MCT is applied. When the ratio becomes close to 1, the algorithm switches to MET. Two thresholds, a low and a high, are set to control the switching.

- *k-Percent best(kPB).* For each job a subset of k % of resources with the minimum execution time is selected. Then, MCT is applied within the

subset. As well as SA this method can be seen as a combination of MET and MCT. While SA is switching between the two, kPB tries to find a trade-off for each decision.

3.4.2 Batch mode heuristics

Batch mode heuristics [136] produce a schedule for a set of jobs (batch) in one scheduler run. Processing multiple jobs in a batch (possibly all the waiting jobs) allows the scheduler to consider more scheduling options. Under such approach, the jobs scheduled before can be also considered for preemption. The downside of such approach is a greater computational complexity and, therefore, a potentially larger scheduling overhead. However, if the use case allows to set large enough intervals between scheduler runs, such increase in solving time is acceptable. The most common batch mode heuristics are:

- *Min-Min.* First, the method calculates the completion time for each job at every resource. Then the job with the earliest possible completion time among all the jobs is scheduled to the corresponding resource. The process is repeated with the remaining jobs and updated ready times of the resources until all the jobs are scheduled. Min-Min allows to minimize flowtime. That drawback of this approach is that the longer jobs get delayed by the shorter ones.

- *Max-Min.* The method is similar to Min-Min. The scheduler compares the jobs by their earliest possible completion times. The job with the latest one is allocated to the corresponding resource. Effectively, this method gives a higher priority to longer jobs.

- *Sufferage.* Allocates the resource to the job which would be disadvantaged the most if allocated to another resource. The sufferage value is calculated as the difference between the second earliest and the earliest completion time of a job. If two jobs compete for the same resource, the job with the greater suffrage receives the allocation. The other job is reconsidered after all pending jobs are analyzed.

- *Relative cost (RC).* Considers both load balancing and minimization of execution times of jobs at resources. Those are conflicting criteria, since for a good load balance the jobs cannot always be assigned to

the fastest resources. In order to find a trade-off, RC uses advanced calculations (see [136]). It computes a *static relative cost* and a *dynamic relative cost* for each pair of a job and a resource. The static relative cost is an execution time of a job at a resource divided by the average execution time of that job over all of the resources. It is calculated once at the beginning of scheduling. The dynamic relative cost is calculated at each iteration and takes ready times of the resources (completion of already scheduled jobs) into account. It is defined as a job's completion time at a resource divided by the average completion time of that job over all of the resources. At each iteration RC analyses every pair of an unscheduled job and a resource. The pair with the smallest product of the static and the dynamic costs is scheduled.

- *Longest Job to Fastest Resource — Shortest Job to Fastest Resource (LJFR-SJFR).* This method is designed as a trade-off between makespan and flowtime minimization. The number of instructions to be executed by each job is assumed to be known in advance. Each resource processes a known constant number of instructions per unit of time. At the beginning of scheduling the resources are idle and the jobs are sorted by number of instructions. Then the principle "longest job to the fastest resource" is applied in such a way that each resource receives one job. After that, the next resource that completes its job is assigned either the shortest (SJFR) or the longest (LJFR) job from the rest of the batch alternately (in rotation).

The immediate and batch mode heuristics for job scheduling are evaluated in simulations in works [135, 136]. The authors conclude that the choice of a scheduling approach depends on the properties of the workload, resources and selected optimization criteria.

3.4.3 Backfilling

There are also methods specifically designed for scheduling of multicore (MPI) jobs. Scheduling of such jobs of varying size potentially leads to resource under-utilization. For instance, the most simple but widely used [129] algorithm is *First Come First Served (FCFS)* [137]. It schedules the first job in the queue each time. If there are not enough resources (cores) to schedule

the first job, the algorithm waits until they become available (resource draining) and so do all the other jobs in the queue. During this waiting, a part of the resources remains idle. In order to utilize CPU time more efficiently, backfilling allows jobs to jump the queue provided they do not delay the first job.

EASY backfilling [138] algorithm calculates the earliest possible start time for the first job in the queue and makes a reservation. Then, it scans the rest of the queue and immediately schedules every job which has enough resources to run and does not interfere with the reservation. Since only the first job gets a reservation in EASY backfilling, the waiting time of other jobs can be significantly prolonged.

The number of reservations can be increased to improve fairness to other jobs. In case of *slack-based* [139] and *selective backfilling* [134] the number of jobs with a reservation is related to their current wait time and slowdown (the ratio of time spent by the job in the system to its execution time) respectively.

Conservative backfilling [134] makes a reservation for every queued job which cannot be executed immediately. It means that backfilling is performed only when it does not delay any previously submitted jobs.

More backfilling algorithms can be found in [140]. However, the main drawback of backfilling algorithms, as other algorithms using reservations, is the assumption of precise knowledge about the job duration and resource performance. In practice, the estimation of job duration, especially provided by users, are far from being precise [141]. Moreover, such estimations may not be available in some systems

3.4.4 Meta-heuristics

Local Search (LS) meta-heuristics [140, 142]. LS explores the search space by applying local changes to the current state (point in the search space) at each iteration until a (sub-)optimal solution is found. A time limit for the search can be explicitly set. LS methods are applied to a broad spectrum of computationally hard optimization problems. In job scheduling, the following methods are often used [123]:

- *Hill Climbing (HC).* Starts at a randomly generated state and moves to the best state in the neighborhood of the current state in each iteration.

If the problem is non-convex, HC may stuck in a local optimum and fail to find the global optimum. This issue can be addressed with multiple restarts of the search. However, the global optimum is not always required in scheduling problems. In practice, it is often preferable to find a "good enough" solution within the given time limit.

- *Simulated Annealing* [143]. In the search process, simulated annealing accepts a worse state with a certain decreasing probability. This allows to escape local optima in the search space.

- *Tabu Search (TS)* [144]. Similar to simulated annealing, TS allows worsening moves, e.g. if no improving moves are available. As a key feature, TS maintains a list of previously visited states in order to avoid cycling.

Population-based meta-heuristics [131]. Population-based approaches maintain and improve multiple candidate solutions using population characteristics to guide the search. The initial population of solutions can be generated either randomly or using the methods described above in this section.

- *Genetic Algorithms (GA)* [145]. In a genetic algorithm (inspired by natural selection) a population of candidate solutions is evolved towards better solutions. At each iteration, the properties of best solutions (according to fitness function) are recombined (with possible random variations) to create a population for the next iteration.

- *Memetic Algorithms (MA)* [146]. Combine the concepts of population-based search and local search. In addition to evolution (similar to GA), a subset of candidate solutions undergoes an individual improvement procedure (similar to LS) at each iteration.

- *Ant Colony Optimization (ACO)* [147]. A probabilistic technique for solving computational problems which can be reduced to finding paths through graphs. The method is inspired by the behavior of ants seeking a path between their colony and a source of food.

- *Particle Swarm Optimization (PSO)* [148]. PSO explores the search space by simulating the movement of a population of candidate solu-

tions (particles) using analogies of position and velocity. Each particle's movement is influenced by its local best known position, but is also guided towards the best known positions in the search-space. An example of job scheduling based on PSO can be found in [149].

There are also schedulers which utilize *machine learning*. The examples are Paragon [150], Quasar [151], and the approach proposed in [152].

3.4.5 Network flows

There are several schedulers using network flow maximization (or min-cost max-flow) [153] as an underlying solving algorithm. Here we would like to discuss the known examples in more detail, since the scheduling approach proposed in this thesis also uses network flows.

A flow network is a directed weighted graph with sinks and sources of flow, where each edge is assigned a cost (per unit of flow) and a capacity (maximum possible flow). The min-cost max-flow problem is to find a maximal flow from source(s) to sink(s) associated with a minimum cost. There is a wide choice of polynomial algorithms solving network flow maximization problem. Formulation of a scheduling problem in form of min-cost max-flow requires several simplification assumptions, and is approached differently by researchers.

According to the authors, Quincy [154] is the first scheduler of computational jobs using min-cost max-flow approach. Both jobs and resources are presented as vertexes in the network. In addition, aggregative vertexes model special scheduling cases when a job remains unscheduled or is ignorant of the choice between a set of resources. The edges of the network represent scheduling options. An edge from a job to a resource means it can be executed there. The capacity of the edges represents the capacity of the resources: each job vertex is a source of a unit flow, and each resource is a sink for as many units of flow as the number of jobs it can accommodate. The cost of the edges expresses scheduling preferences: data locality, cost of preemption, priority of jobs, etc. When the min-cost max-flow problem is solved, the resulting flow presents the scheduling decisions: a unit flow passing through a job shows to which resource it is scheduled. The complexity of such problem depends on both the number of jobs and the number of resources. Also, Quincy uses a complex algorithm for network construction at each scheduling iteration,

which involves communication with every resource in the system. In case of modern large-scale and heavily loaded systems the scheduling overhead may significantly increase and become a bottleneck. Experiments show that it can reach tens of seconds for the cluster of size of 10,000 machines. Quincy was developed by Microsoft and is used in practice as a scheduler for Dryad [75] framework.

The Firmament [121] scheduler uses very similar problem formulation but addresses the issue of scheduling overhead. It uses several solvers with distinct (adjusted for the problem) min-cost max-flow algorithms running in parallel. The solution is taken from the scheduler that finishes first. Such approach allowed to significantly improve worst-case performance. The equivalence classes are used to reduce the number of edges in the scheduling network and thus decrease the problem complexity. Also, the Firmament uses incremental updates to the network state. The evaluation was performed using simulations and in a testbed cluster of 40 machines. The authors claim that the scheduling latency is reduced by the factor of 20 compared to Quincy. Firmament is an open-source software complemented with its own cluster management system. There is an ongoing project for integration of the Firmament scheduler into one of the most popular cluster management systems Kubernetes [155].

4 Optimization of data access

In the previous section we have already discussed various optimization goals of job scheduling. Here we focus on a specific one — data access. Efficient data access for data-intensive distributed computations has become an important optimization problem with the establishment of big data paradigm. In modern systems such optimization goes beyond job scheduling alone. From an eagle's eye view it can be seen as a union of multiple aspects: workload mapping to resources, CPU scheduling, data transfer and prestaging, data stream routing, network scheduling, storage management and data replication. In fact, each of these aspects represents a distinct sub-domain. Each of the named aspects is a complex optimization problem on its own, e.g. CPU scheduling and optimal data placement are NP-complete problems [156]. Therefore, each practical solution considers one or a combination of those aspects. The end-to-end optimization can be seen as a composition of multiple applied solutions each responsible for its part of a global problem. Large-scale computing systems (e.g. scientific grids) deploy a customized stack of components to enable distributed computing. The division of duties between components can vary and overlap in particular implementations. In a very general case, aside from (a) job scheduling / workflow management data access is optimized by components that provide (b) data placement / prestaging, (c) data replication and (d) network load balancing. Below we would like to summarize their general optimization efforts with respect to data-intensive distributed computing.

4.1 Data aware job scheduling

Workload Management (and job scheduling) System (WMS) among other functionalities performs mapping of jobs to resources. We already presented various examples of jobs schedulers and WMS in Section 3.2. WMS may also create dedicated data transfer jobs which are either treated as "normal" jobs (scheduled along with computational jobs) or submitted to a dedicated data placement service. In both cases, a data transfer job specifies the data to be transferred and the destination. It may also specify current data hosts, the deadline or the precedence relation with the corresponding computational job.

Data awareness gets addressed increasingly often in more recent works (21 % of most cited papers over decade [102]). The general methods for scheduling parallel jobs with communication delays are extensively described in [157] and [158]. Data access overheads can be added to estimated job execution time to provide better placement decisions by the scheduling methods described in Section 3.4 and their derivatives such as XSufferage [141] and Storage Affinity [159]. Many scheduling algorithms applied in practice exploit *spatial data locality*, when jobs are scheduled to the nodes holding the data or as close as possible in a network sense. This can be implemented either as a hard or a soft constraint. For example, there are algorithms which utilize cost function for multi-criteria optimization and assign a higher cost to job placement decisions which imply remote data access [121, 154]. To exploit *temporal data locality*, the jobs sharing the data are placed to the same resource, so that the number of data transfers can be minimized. Job scheduling approaches applied in practice delegate data transfer scheduling and data placement optimization to external components. The job allocation decisions taken by the scheduler limit the options for future transfer routing and scheduling. Since the job scheduling is disjoint from transfer scheduling on the network topology, concurrent data transfers for newly allocated jobs may lead to network congestion in the case when bottlenecks exist.

4.1.1 Examples

The Adaptive Regression Method (AdRM) [160] is focused on predicting the performance of data transfer operations in network-bound distributed data-intensive applications in a multiuser grid environment. As stated by the authors, those predictions can be used for comparison of candidate schedules of computational jobs, however, an exact scheduling algorithm is not presented in that work.

DENS (data center energy-efficient network-aware scheduling) [161] is particularly relevant in data centers running data-intensive jobs with low computational load, but produce heavy data streams directed to the end-users (such as video streaming and geographic information systems). The design and specification are tight to the underlying data center architecture: in presented work it assumes three-tier fat tree architectures (most common for data centers), but is extensible to other cases. The observed load of con-

sidered systems is close to 30 % on the average. The main idea of optimization is to aggregate the load on a subset of machines while not compromising network performance. This allows to shutdown the rest of resources (machines and network switches) in order to reduce energy consumption.

In [162] authors considered a problem of co-scheduling of job allocation and data replication. The considered infrastructure resembles a computational cluster connected to grid. It consists of heterogeneous computing nodes with local stores connected over LAN. The input data are placed at external stores connected over WAN, where a bottleneck bandwidth to each store is known. The scheduler based on genetic algorithm defines job order, job assignment to nodes and data assignment to local stores in order to enable backfilling, data prestaging and temporal locality. An exact duration of jobs is assumed to be known in advance. Also, neither storage nor network sharing is considered.

An explicit model distributing jobs over a hierarchical grid with respect to the network bandwidth was proposed in [163]. The network structure of the grid was modeled as a tree and all the files were assumed to be of the same size and processing duration. With the constantly growing number of participating sites the hierarchical structure of the grids (e.g. WLCG) is not strongly held. Moreover, the limitations of a hierarchical topology for data dissemination were demonstrated in works [124, 164, 165].

4.2 Data transfer and placement

Data Placement Service (DPS) or Distributed Data Management (DDM) system (e.g. Don Quijote 2 [166], Kangaroo [167], Kepler [168], Pegasus [169], PhEDEx [170], Rucio [171], Stork [2]) executes data transfer requests submitted by WMS, computing jobs or users. A general example of workflow decomposed into computational and data placement jobs, given in [2], is pictured in Figure 2. If transfer requests are submitted in advance it allows to perform data prestaging prior to computations. Also, output data can be staged out for transfer from processing nodes to its destination. Staging data in and out allows to overlap data transfer with computation in order to improve overall processing throughput.

Bharathi and Chervenak [10] have studied data staging strategies in data grids and their impact on the execution of scientific workloads. They clas-

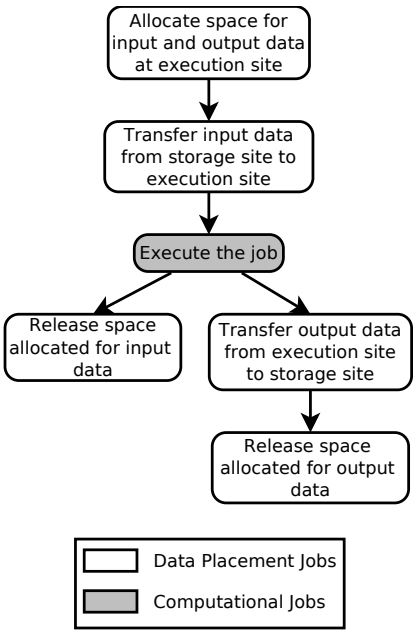


Figure 2: Computational workflow decomposed into computational and data placement jobs [2].

sified staging strategies performed by DPS into decoupled, loosely coupled and tightly coupled based on interaction with job scheduling system. In the *decoupled* data staging mode, data placement operations are carried with minimal interaction with WMS. DPS is supplied with a list of data items and a deadline by which they need to be staged into different computer resources (sites). This is typically the case when users provision or reserve compute and storage resources at sites.

In the *loosely-coupled* staging mode the workflow manager submits transfer jobs to the DPS. To take the advantage of parallelism in the workflow each computational job should be preceded by a transfer job that stages-in the data required by that job. However, this comes at a cost, as each transfer job may be processed by all or most of the workflow execution components, leading to significant overheads in the execution of a data-intensive workflow. Further, data movement services may be subjected to high loads if the workflow manager releases many transfer jobs concurrently.

In (prospective) *tightly coupled* staging mode the workflow manager is integrated with data placement service. The authors envision such integra-

tion as a single logical entity that releases both computational jobs and data transfer jobs in a coordinated manner. The resulting scheduler could alter the order of release of computational jobs based on data availability and constraints on storage resources and data management services. Such scheduler would be able to identify independent paths of execution in the workflow and release computational jobs in a manner that distributes data transfer load across the entire system. The authors also emphasize the importance of appropriate data staging in and out of storage-constrained resources, since data-intensive workloads consuming and generating large amounts of data may affect other jobs utilizing the same shared file system. Finding a valid schedule that allows execution of a workflow with storage constraints is an NP-complete problem. The authors have performed simulations of these data staging strategies using traces of different scientific workflows (such as LIGO, CyberShake and Montage). The results showed that the best choice of staging mode depends on the workflow properties. In short, the strategy which provides the smallest interference of data transfer with job execution is preferred. The workflows which stage-in large amounts of data at initial stages would benefit from decoupled transfers. The loosely-coupled scheme is beneficial when fewer transfer jobs are created and there is a greater overlap in data transfer and execution of computational jobs. The tightly-coupled strategy was tested to optimize for a constrained storage. The resulting optimization appeared efficient for certain types of workloads only.

DPS addresses transfer failures and provides multiple optimizations for data transfer performance. It selects an appropriate transfer protocol and tunes transfer parameters (e.g. number of streams, block size) based on performance monitoring [172, 173]. Advance storage reservation, when supported, allows to avoid transfer failures due to insufficient space. DPS typically implements concurrency control by limiting the number of active transfers between any pair of hosts or in the entire system. To our knowledge, the underlying network topology and per link bandwidth scheduling are not typically considered by DPS. Replica selection strategy is applied when the requested data are present at multiple locations. It allows to balance network and server load. In various implementations the replica selection is performed either by WMS or DPS. Simple policies are the most commonly applied: random, explicitly specified list of preferences [169], network hierarchy (closest

parent/child of the destination node), best connectivity to the destination (e.g. by latency, bandwidth, IP address driven) or least loaded host. More advanced replica selection policies which utilize logs of previous performance, transfer probing and multi-criteria optimization are surveyed in [164]. It is important to note that the DPS has a limited number of options for the network load optimization, since the transfer destinations and the deadlines (or order) are determined by the allocation of computational jobs by WMS. It is a natural choice to prioritize CPU scheduling over network and storage in CPU bounded systems. However, such an approach may cause problems in network-bounded cases, e.g. when data-intensive applications are executed outside of a dedicated high-end network infrastructure.

Staging out the output data of data-intensive jobs is considered in [167] and [174]. In both approaches the output is handled by a dedicated service asynchronously to computations. In [167] the data is buffered at intermediate nodes along the transfer path in order to mitigate network failures or congestion.

4.3 Network usage optimization

Several network technologies have a high potential for optimization of data access for distributed data-intensive applications [9]. Dynamic Circuit Provisioning (DCP) allows to set up circuits on demand for high priority transfers and provide bandwidth guarantees, bypassing a possibly busy default routing path [175]. Software Defined Networking (SDN) [176, 177] enables centralized control over the routing of data streams. It can be utilized for multipath load balancing (e.g.[178, 179]) and scheduling of concurrent network flows using a detailed network representation. For example, the problem of network scheduling is addressed in studies [180, 181, 182]. The research is ongoing for closer integration of DCP and SDN technologies with WMS, DPS and monitoring systems used in practice, such as AliEn [183], MonALISA [184], PanDA [185] and PhEDEx [170]. In general, multipath load balancing and scheduling are applied to active data streams or to transfer requests submitted after the computational jobs had been scheduled. It would be practical, if detection of bottlenecks at the transfer scheduling phase would allow to reconsider (or influence) job allocation.

An emerging concept of Information Centric Networking [186, 187] en-

ables efficient sharing of bandwidth and storage by multiple users (or jobs). To a certain extent, it can be seen as an integration of data replication into networking. However, ICN does not meet the use cases with low data reusage between parallel computations.

In previous collaboration between BNL and NPI CAS, the problem of efficient data transfer in a grid environment was addressed [11]. Data transfers between n computational sites and m data locations were considered but job scheduling was not covered by that work.

4.4 Data replication

Big data computing strongly relies on data replication strategies for data dissemination, optimization of access, fault tolerance and safety against loss. Having more replicas of the same data across the systems gives more job scheduling options to WMS exploiting data locality. The replication strategies also exploit temporal and local correlations in data requests. Such approaches were efficiently applied to many workflows and systems and were adjusted to their specifics. Data replication is used in various distributed contexts: data grids, clouds, distributed databases, content distribution systems, mobile systems, storage technologies and etc. Here we imply the context of large-scale distributed computing systems for scientific applications such as grid, however, the general observations can be related to other contexts.

The idealistic solution would be to replicate all data to all possible locations (e.g. every site in grid), so that every instance could access data locally. Obviously, such solution is not realistic since the storage space at sites and network bandwidth between them are limited. Therefore, a data replication strategy has to answer the following questions:

- When to create/delete replicas?
- Which data should be replicated?
- Which replicas should be deleted?
- Replica placement: where to place the replica?
- Replica selection: where to access the data when multiple replicas exist?

4.4.1 Replica placement

Optimal replica placement is an NP-complete problem [188]. A great variety of replication strategies exist. Each implementation targets a particular combination of infrastructure and workflow (data access pattern). In general, the replication strategies can be classified using the following criteria:

Static vs. Dynamic. In static replication all decisions are made before the system is operational and not changed. It is a viable choice for a well-defined non-changing environment and data access pattern. Dynamic replication adapts to the observed access pattern, performance of the system components and their load. Such flexibility is necessary for modern large-scale systems.

System topology. Each replication strategy is designed with some target system topology in mind (e.g. grid topologies summarized in Section 3.1). Simulations of replication strategies in various grid architectures [165] have shown that architectures with less restrictive communication and having multiple network routes between sites provide more data access optimizations. In such architectures the response time is decreased, regardless of the applied replication strategy. Many authors regard arbitrary graphs as the most realistic and interesting representation of data grids for replication problems.

Centralized vs. decentralized. Centralized replication strategies rely on a central authority to control all aspects of data replication. A centralized system has one master replica for each data instance which is updated and the updates are propagated to other replicas. In a decentralized approach, multiple entities (such as grid sites or users) can make decisions about replication. Additional synchronization is needed to maintain data consistency in case of updates. In many big data applications, including scientific computing, read-only data access is used in data processing. Therefore, most of the common replication strategies do not consider consistency on updates.

Replica granularity. Corresponds to the unit of data that can be copied independently. It may be individual files, fragments of files (objects, blocks) or sets of files (datasets)[189]. Most of the replication

strategies for scientific applications in grids consider file level of replication.

Storage integration. *Tightly-coupled* replication mechanisms are tied to storage architecture on which they are implemented including file system and I/O mechanism. Since replication is implemented at a low level, it is often invisible to high-level applications and users. *Loosely-coupled* replication is superimposed over the existing file systems and storage systems. Such mechanisms interact with the storage systems through standard file transfer protocols and at a high level. Replication can be initiated and managed by applications and users. Intermediate systems exert control over the replication mechanism but not over storage resources [124].

Push vs. pull. A replication event can be triggered differently. A file can be replicated to a node upon request from that node (PULL). Client-side caching is also regarded as pull replication. In push-based approach a replication is triggered externally from the destination node, e.g. by central service or initial replica host. Push-based replication is utilized in proactive approaches which try to optimize for future data requests. Pull-based replication can be regarded as a reactive approach which adapts to current access pattern [165].

Periodicity. The replication can be adjusted continuously, where actions are taken in response to every file request or other events. Alternatively, replication can be performed periodically, e.g. triggered after every fixed time interval [190].

Degree of replication. The number of replicas can be fixed for every item, or dynamically adjusted based on its importance and popularity.

Considered parameters. The parameters considered by replication strategies can be divided into two types: data associated and system associated. Data associated parameters most commonly measure "popularity" of a given file, e.g. number of requests per file or time elapsed since the last request. Such parameters also may include file size, type, grouping of files (clustering) and etc. A similarity of parameters used by replication strategies and caching policies demonstrates the close

relation of those fields. System associated parameters may include available storage at sites, server load, network bandwidth and clustering. There are replication strategies which utilize information about previous requests and system performance (e.g. logs and monitoring data) to predict future behavior [190].

Objective. Since data replication is often approached as an optimization problem various objective functions are used. Similar functions can be utilized as metrics to compare distinct strategies. The objectives typically reduce: job execution time, response time on file request, bandwidth consumption, access latency, local miss ratio, utilized storage. Custom metrics are used to evaluate data locality, quality of replica distribution, optimality of replication degree. In addition, when usage of resources is charged (e.g. commercial clouds) the involved cost (for storage and network usage) can also be considered by replication strategy.

4.4.2 Replica selection

Replica selection is one of the crucial aspects of data management in big data applications. When a requested file is replicated to multiple hosts the replica selection policy determines which one should be selected. In hierarchical replication strategies, such choice is driven by the topology: the closest parent (or child) node is selected. More general algorithms typically rely on one of the following parameters, or their combination [164]:

- Best bandwidth between the host and destination.
- Round trip time (RTT).
- Network distance, e.g. number of hops.
- Response time. History of previous transfers between hosts can be used to estimate response time for future transfers.
- Load of the host, e.g. number of active requests.
- Access cost.

In addition, custom metrics can rank hosts by security and fault tolerance. Before transferring large volumes of data an algorithm may probe the hosts by fetching a small chunk of data. Advanced techniques are applied in order to optimize for multiple parameters, including clustering analysis, ant colony optimization, auction protocols and neural networks [164].

To our observation, common replica selection algorithms apply a greedy approach, i.e., the best replica host is selected for each request but their interference is not considered. While the current network and server load is used in decision making, the influence of newly placed request falls out of the picture. Therefore, the concurrency control, if supported, is delegated to a data placement service or transfer tools. Alternatively, Zerola et. al. [11] proposed a transfer approach which optimizes for parallel transfers between n sources and m destinations. The approach allows to route and schedule the transfers over the network in order to avoid congestion and minimize response time. The requests for the same file from distinct users are grouped together and served in such a way that the combined number of hops over the network is minimized. The file is transferred just once over the shared links on the paths to multiple destinations (transfer overlap). It allows to reduce bandwidth consumption at the shared network links.

A general picture of the interaction between data replication, data transfer and job scheduling in data grids is described in [124]. The impact of the system topology on efficiency of common replication strategies is studied in [165]. The roles and objective functions of replication strategies in data grids are surveyed in [189]. The survey [164] provides an insight into replica selection strategies. The replication strategies which utilize data mining techniques, i.e., analyze data on previous usage and performance in order to optimize for future, are compared in [190]. Optimization of data-intensive applications in grid was studied in [191]. In that work, an optimization was achieved by replication of highly used files to more sites while the jobs were executed where their input data is located. Similarly, the Storage Affinity [159] approach exploits data re-utilization to improve the performance of an application in grid. Both approaches are classical examples of techniques improving data locality, and have found a wide application in practice.

To summarize, modern big data computing strongly relies on data replication strategies for data dissemination and optimization of access. Those

replication strategies exploit temporal and local correlations in data requests. Such approaches were efficiently applied to most workflows and systems and were adjusted to their specifics. However, there are workflows, such as data production (discussed in Section 5) where each file is accessed just once. In such cases the common replication strategies cannot be efficiently applied.

5 Computing in High Energy and Nuclear Physics

Data processing in modern High Energy and Nuclear Physics (HENP) is a prominent example of big data applications. This scientific field was among the pioneers who encountered challenges of big data even before the concept became established and spread to other scientific fields and industries. In 2017 the amount of permanently archived data of all the experiments at Large Hadron Collider (LHC) at CERN has reached 200 petabytes [192]. The same year the scientific data and computing center at Brookhaven National Laboratory (BNL) hosting the STAR experiment reached 100 petabytes of recorded data [193].

In HENP experiments a beam of particles (e.g. electrons, protons, antiprotons or heavy ions) is accelerated close to the speed of light. The beam collides into another one accelerated in the opposite direction or into a fixed target. The collision takes place inside a detector which measures the outcome of particle interactions. The further sophisticated analysis of the collected data allows to discover new particles, such as a Higgs boson, and study fundamental properties of matter and physical interactions.

5.1 Computing activities

Modern particle accelerators operate at enormously high energy and luminosity in order to provide sufficient event rates in the detectors. Such event rates are necessary to accumulate enough statistics for scientific analysis within a reasonable time period. For instance, the STAR experiment at RHIC detects events at a rate equivalent to 70 GB/s. The full detector rate of ATLAS and CMS experiments at LHC approaches 1 PB/s [7]. The raw data from the detector are filtered by a trigger system and reconstructed into physical events which are further analyzed by scientists. In addition to that, computer simulations of the experiment are performed in order to compare theoretical predictions against the real data. In general, four major types of computing activities can be distinguished in HENP:

Data acquisition. Since the goal of a HENP experiment is to discover new physical phenomena, not every registered event is of scientific inter-

est. In fact, most of the events correspond to well studied phenomena and are not recorded. A trigger system of a detector filters events based on fast analysis of the signal. It may consist of multiple levels: the higher the level – the more detailed analysis and the smaller the data flow. Typically the level 1 trigger runs on highly specialized hardware installed at the detector. It operates at frequency tied to beam crossing rate ($\sim$ 10 MHz) and each filtering decision is performed within tens of nanoseconds. Triggers of higher level may operate on general computing hardware asynchronously to data readings. The data rate after the trigger system depends on the specifics of an experiment. For example, it is up to 400 MB/s for STAR at RHIC, 200 MB/s for LHCb, around 1 GB/s for ATLAS and CMS and 4-5 GB/s for ALICE at LHC [7]. The data that survived all the trigger levels are recorded to a permanent storage in form of files. Each file contains event records taken under the same experiment setup. A complete set of files recorded under the same experiment conditions is called a *dataset*.

Data production. Raw data from the detector describe its state at each recorded event. It consists of readings of currents and voltages at many elements of the detector (around 150 million of electronic channels). Before the data can be analyzed by scientists, it is necessary to calibrate the measurements and reconstruct tracks and properties of the particles. This process is called data production. It is also often referred as reconstruction or, in a more general big data terminology, preprocessing. The data production is performed by campaigns where a recently taken dataset is processed. Each file is processed exactly once within one campaign. Such campaign typically lasts for several months and processes hundreds of terabytes of data. Sometimes, a campaign is repeated (after a significant time) when it allows to improve the quality of the reconstructed data. The reconstructed data can be effectively utilized only after an entire campaign is finished. For this reason, it is highly desirable to execute data production with the shortest possible makespan on given resources. The data production is typically centrally submitted and managed by computing administrators.

Simulation. Monte Carlo simulations [4] are used to recreate the

physical events and the response of the detector. The simulations are based on theoretical models and the results are compared to experimental data at later analysis stages. The simulations are CPU-bound applications which do not require significant amount of input data but produce large amounts of output. Overall, the simulations often consume a major fraction of CPU resources of a HENP experiment. This type of computations is also centrally managed.

User analysis. The users (scientists and scientific groups) perform analysis of reconstructed data from the detector and simulations for new scientific discoveries. Data access patterns of such computations are less predictable: the users are interested in various (overlapping) subsets of data and perform analysis independently. Each particular file can be processed many times by distinct users. Users may download the required data to a particular facility available to them and perform the analysis locally. More commonly, they may submit their computational jobs to the central scheduling system, specify the dataset to be processed and the destination for the output. The central scheduling system allocates the jobs according to its policy, user preferences and spatial/temporal data locality. In general, the same optimization techniques are applied as for scheduling of other common big data applications.

Data level parallelism is typical for HENP workload. This means that the computations are divided into independent jobs, each job processes/produces its portion of data (a file or a set of files). Except for the user analysis, those portions of data do not typically overlap. All types of computations are I/O and CPU intensive, while the simulations are CPU bound.

Petabytes of data being collected and processed by HENP experiments annually require vast amounts of computing, storage and network resources. Therefore, these large-scale experiments rely on distributed computing and utilize computing facilities at many geographical locations. Aggregation of such facilities to address a common computing problem is called a grid. There exist multiple grid projects, often tied to specific scientific collaborations. Worldwide LHC Computing Grid (WLCG) [3, 4] is a global collaboration responsible for building and maintaining a data-oriented infrastructure required by the experiments at the LHC. The WLCG cooperates with several

grid projects such as the European Grid Infrastructure (EGI) [96] and Open Science Grid (OSG) [95].

5.2 Tiers

One important property of HENP computations is that the data originated from a single source (the detector) is being propagated (replicated) to many remote computing facilities. This property guides the design and operation of the corresponding computing models. At glance, the initial idea was to follow a hierarchical (tree) structure: the detector facility is a root of a tree, regional centers are the next level and so on. Each site was assumed to access (and sometimes store) a subset of the data from its parent. However, such model was proven to be overly restrictive. As a result, the current concept of Tiers in WLCG provides more flexibility for site roles and communication. It defines four types (Tiers) of sites (see Figure 8):

Tier-0. A logically unique Tier-0 function is performed by two physical sites: one is the CERN Data Center in Geneva (Switzerland) and the other is located at the Wigner Research Center for Physics in Budapest (Hungary). These sites are connected with two 100 GB/s data links with a latency of 30–35 ms for fast synchronization. Tier-0 is responsible for data acquisition, reconstruction, archiving to a tape storage, and for the distribution to Tier-1 sites. All the data generated by the experiment (raw, reconstructed, analyzed and simulated) is permanently stored at the tape archive of Tier-0 and in two more copies at distinct Tier-1s.

Tier-1. There are 13 LHC Tier-1 sites, which can be seen as regional centers. They are exploited for large-scale, centrally-organized activities and can exchange data between them and any of the Tier-2 sites. They are responsible for storing raw and reconstructed data, data production, simulation and safe-keeping of important analysis output. Also, they may provide capacity for user analysis jobs. Having a high-speed network connection to Tier-0 is essential for such site. Also, a Tier-1 must have a tape archive for a permanent storage of experimental data.

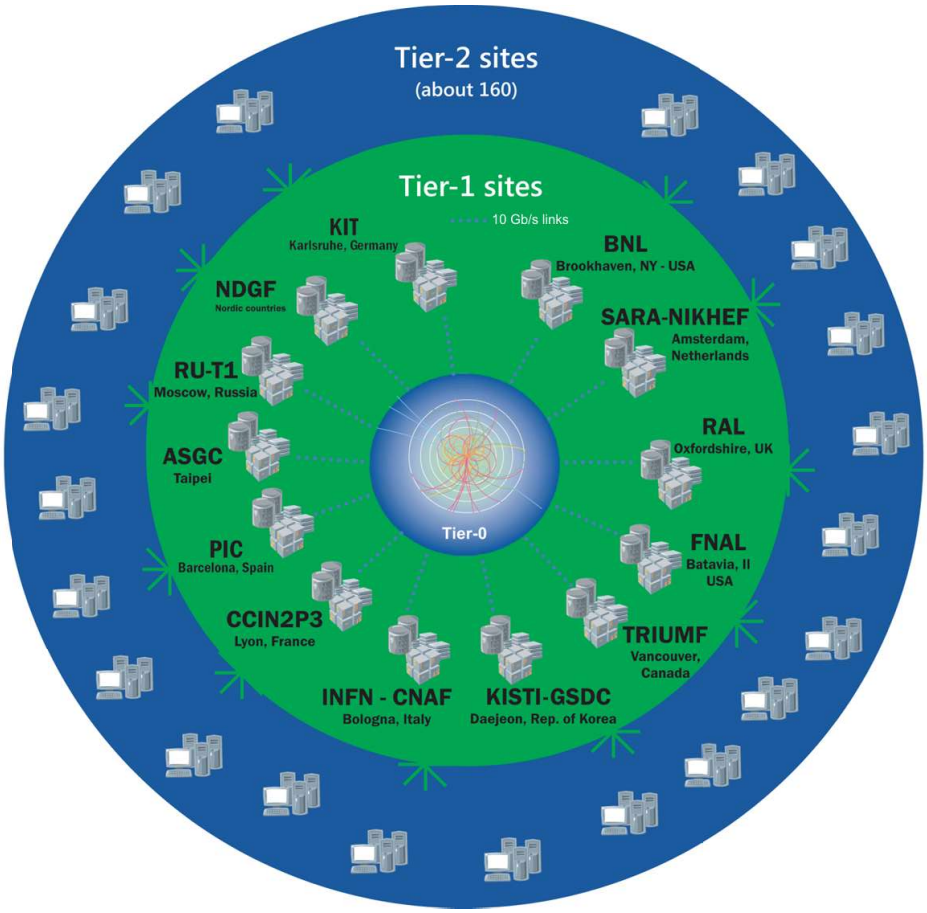


Figure 3: Tier structure of WLCG [3].

Tier-2. There are about 160 Tier-2 sites in WLCG placed around the world. Typically, those sites are large computing facilities of scientific institutes and universities. Despite a smaller size, the aggregated CPU and storage capacity of Tier-2s exceed those of other tiers. The primary purpose of those sites is running user analysis jobs. Tier-2s do not have tape archiving and do not provide long-term custodial storage. However, data production and simulation can also be performed there upon available capacity. Setting-up and running a Tier-2 site for WLCG in Prague is described in [15].

Tier-3. It is the most flexible Tier level as there is no formal agreement between WLCG and Tier-3s on their respective roles. It can be, for

example, a computer server of a scientific group, faculty or department. Such flexibility allows facilities to temporally pledge available capacities or to perform specific tasks.

In addition to the established primary infrastructure, HENP experiments tend to utilize external volatile/elastic resources [194]. Examples of such resources are: scientific and commercial clouds, volunteering computing, unused capacities of other experiments/collaborations.

Each HENP experiment maintains a separate software/middleware stack to enable the computations. However, the research community tends to join the effort, when possible, to develop versatile and solid solutions. For example, such software as root [195], GEANT [196], PanDA [185], DPM [197] are used across many experiments. Also, general concepts and approaches are often shared, e.g. virtualization, pilot jobs, data trains. When it meets the criteria, the experiments utilize common grid middleware (e.g. Condor [80], GridFTP [198]) and contribute to the development.

6 Study of distributed job and data transfer scheduling using constraint programming

In previous collaborative work between BNL and NPI CAS, a new data transfer tool was developed [11]. The tool allows to optimize data transfer across a distributed system with the help of centralized planning based on Constraint Programming (CP). As the research has shown, the global planning of data transfers across a data grid can outperform widely used heuristics such as Peer-to-Peer [199] and Fastest link [190]. Typically, data transfer and distribution is not a stand-alone problem and should be viewed in a broader context of data processing. However, the approach was focused on planning of data transfers over network, but did not consider CPU and storage allocation to processing jobs. Since any of the resources (CPU, storage and network) may become a bottleneck and decrease resulting processing throughput it is desirable to include all of them into the scheduling problem. For this reason, in this study we reformulate the constraint programming model from that research to provide joint scheduling for CPU allocation, data transfer and file placement at storage. The optimization is achieved by ensuring that none of the resources is overloaded at any moment of time and either (a) input data transfer for each job is performed in advance before its start or (b) the jobs are scheduled where the data are already present.

Problems of scheduling, planning and optimization are being commonly solved with the help of Constraint Programming (CP) [200]. It is a form of declarative programming which is widely used in scheduling, logistics, network planning, vehicle routing, production optimization, etc. Here we introduce our constraint satisfaction problem formulation for distributed data processing and test it in simulations.

A Constraint Satisfaction Problem (CSP) consists of domain variables, domains (a set of possible values of a variable) and constraints in form of mathematical expressions over variables. A solution to CSP is a complete assignment of values to variables which satisfies all the constraints. An optimal solution is the one with the minimal/maximal value of an objective function of variables.

Our main goal here is to study if the scheduling of multiple resources can be efficiently addressed as a Constraint Satisfaction Problem (CSP) in case

of distributed computing; and if such scheduling can provide an advantage compared to traditional techniques of the field (independent management of the resources). To answer those questions, in this chapter we present our constraint based scheduling approach and simulations of distributed data production. In the simulations we consider a simplified use case derived from remote data production of the STAR experiment and compare performance of our approach to other common scheduling policies.

6.1 Model and solution overview

The following input parameters are necessary to define our CSP:

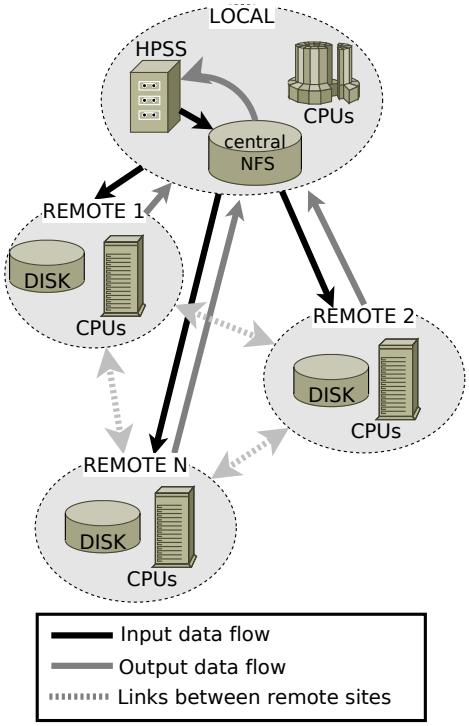


Figure 4: Resources considered in CSP based approach.

Computational grid (see Figure 4) is represented by a directed weighted graph where nodes are computational sites c_i with a given number of CPUs $NCPU_i$ and storage space $Disk_i$; edges are network links l with weight o_l, called slowdown, which is the time required to transfer a unit of data ($o_l = \frac{1}{b_l}$, where b_l is the bandwidth of the link). A set

of incoming links of a site is denoted L_i^{to}, a set of its outgoing links is L_i^{from}. A dedicated storage-only facility can also be modeled as a site with $NCPU_i = 0$.

Set of jobs. Each job j has a duration p_j, it processes one input file f_j^{in} and produces one output file f_j^{out}. The input file is initially placed at each of source sites $Sources_j$ and output file must be transferred to one of destination sites $Dest_j$. Size of a file f is denoted $size_f$.

Our goal is to create a schedule of jobs at computational sites, transfers over links and placement of files at storages for a given computational grid and a set of jobs. In order to solve this problem the variables of our model define the *resource selection* and *timing* of each task:

Resource selection variables define a site where the job j will be executed and a transfer path for each file f (either input or output of a job). A set of boolean variables Y_{ji} defines if a job j is executed at a site c_i (either 1 if *true* or 0 if *false*). Similarly, the transfer path is described by a set of boolean variables X_{fl} where 1 means that a file f will be transferred over a link l and 0 means the opposite.

Time variables are: S_j^J is a start time of a job j, S_{fl}^T is a start time of a transfer of a file f over a link l, S_{fi}^F is a start time of placement of a file f at a site c_i and p_{fi}^F is its duration.

6.1.1 Model assumptions

Two important assumptions which are reused in the current model were proven in a work on planning of data transfer in grid [11].

The first assumption states that the entire set of jobs (queue) can be incrementally scheduled by subsets (chunks) without significant loss of optimality. Such an approach helps to reduce the search space and thus improve solving performance. Moreover, planning for shorter periods and more frequent generation of plans (or re-planning) provides a better level of adaptability to the changing environment.

The second assumption states that a network link can be modeled as an unary resource without loss of generality. In other words, in our model we consider that only one file can be transferred over a link at a time. The

measurements in [11] have shown, that a sequential transfer of a set of files (using multiple threads for each transfer) does not require more time than a parallel transfer of the same set of files over the same link.

6.1.2 Solution overview

Computational time is an important factor for online scheduling. In case of scheduling of a large number of jobs by portions, fast performance of a scheduler is required to produce an entire schedule within reasonable time. Also, if a schedule has to be reconsidered due to system reconfiguration or execution failure a new schedule should be generated fast. An incomplete search with an explicitly set timeout is used in our approach in order to reduce solving time. It allows to find a suboptimal solution of required quality within a given time limit. Also, in order to reduce an overall problem complexity, the problem is divided into two subproblems and the search is performed in two stages:

1. Planning stage: instantiate the resource selection variables in order to assign resources (computational sites and links) for each task (computational jobs and file transfers). This stage includes:

    - Constraints on file transfer paths and job allocation.
    - Estimation of the makespan T_{est} for a given resource assignment.
    - Search for an assignment of X_{fl} and Y_{ji} which minimizes estimated makespan.

2. Scheduling stage: instantiate time variables in order to define a start time for each task. This stage includes:

    - Fixed values of X_{fl} and Y_{ji}.
    - Constraints on the order and duration of tasks.
    - Cumulative constraints on resources.
    - Search for an assignment of S_j^J, S_{fl}^T, S_{fi}^F and p_{fi}^F with a minimal makespan.

6.1.3 Constraints at the planning stage

At the planning stage the problem is to assign tasks (computational jobs and file transfers) to resources (computational sites and links) in such a way that the set of tasks could be completed within the minimal time. In other words, at this stage we instantiate resource selection variables X_{fl} and Y_{ji} minimizing estimated makespan T_{est}. It considers estimated completion time among all resources which is the time required to process the tasks assigned to them. The optimization function at this stage is not expected to provide correct makespan estimation, but rather to balance the load among the resources. It is defined as a sum of the maximal estimated completion time among the sites T_i^{CPU} and among the links T_l^{link}:

$$T_{est} = \max_{i \in Sites}(T_i^{CPU}) + \max_{l \in Links}(T_l^{link}) \tag{1}$$

To find the estimated completion time for each site T_i^{CPU} we divide the total duration of assigned jobs by the number of CPUs and compare it to the duration of the longest assigned job. The largest value is selected as an estimation:

$$T_i^{CPU} = \max\big(\frac{\sum_{j \in Jobs} Y_{ji} \cdot p_j}{NCPU_i}, \max_{j \in Jobs}(Y_{ji} \cdot p_j)\big) \tag{2}$$

The estimated completion time of link l is defined as follows:

$$T_l^{link} = \sum_{f \in Files} X_{fl} \cdot size_f \cdot o_l \tag{3}$$

Figure 5 illustrates a transfer path assigned to a single job j at the planning stage. The sites which are input sources for this job are colored red, the output destinations are colored blue, the site selected for the job execution is colored green, the white sites are either intermediate or not used. Red arrows are the links selected for input transfer, blue arrows are the links selected for output transfer, unused links are gray dotted lines. Numbers inside the sites indicate their id. The small red and blue numbers near links are the values of X_{fl} for the input and the output file, respectively. The small green numbers near sites are the values of Y_{ji}. The gray circles and letters refer to the constraints described in the following text.

For each job we have to assign a transfer path for an input and an output file which can be defined by the following constraints:

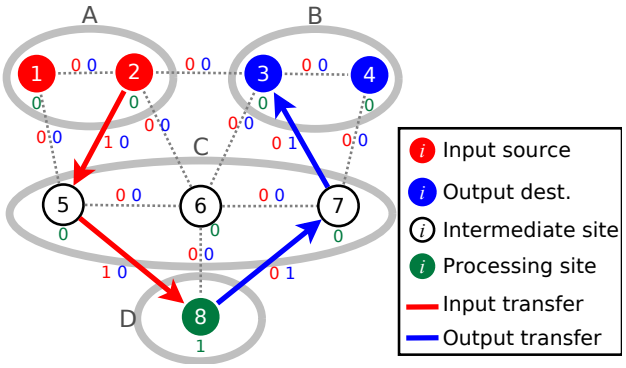


Figure 5: Example of a transfer path for one job.

- An input file has to be transferred from one of its sources over exactly one link (see A in Figure 5). The file should not be transferred to a site which already contains it.

$$\begin{gathered} \forall j, f = f_j^{in} : \\ \sum_{l \in A_1} X_{fl} = 1, \quad A_1 = \bigcup_{c_i \in Sources_j} L_i^{from}; \\ \sum_{l \in A_2} X_{fl} = 0, \quad A_2 = \bigcup_{c_i \in Sources_j} L_i^{to}; \end{gathered} \tag{4}$$

  here A_1 and A_2 are the sets of links leading from/to all the source nodes of file f.

- An output file has to be transferred to one of its destinations over exactly one link (see B in Figure 5). The file should not be transferred out of its destination.

$$\begin{gathered} \forall j, f = f_j^{out} : \\ \sum_{l \in A_3} X_{fl} = 1, \quad A_3 = \bigcup_{c_i \in Dest_j} L_i^{to}; \\ \sum_{l \in A_4} X_{fl} = 0, \quad A_4 = \bigcup_{c_i \in Dest_j} L_i^{from}; \end{gathered} \tag{5}$$

  here A_3 and A_4 are the sets of links leading from/to all the destination nodes of file f.

- If an intermediate site c_i (neither source, destination nor selected for the job execution) has an incoming transfer of a file f it also has an

outgoing transfer of the same file (see C in Figure 5):

$$\sum_{l \in L_i^{to}} X_{fl} = \sum_{l \in L_i^{from}} X_{fl}. \tag{6}$$

- There must exist exactly one incoming transfer of an input file f_j^{in} and exactly one outgoing transfer of an output file f_j^{out} at the site which was selected for the job execution ($Y_{ji} = 1$) (see D in Figure 5).

$$\begin{gathered} \forall j, i | Y_{ji} = 1 : \\ \sum_{l \in L_i^{to}} X_{f_j^{in} l} = 1; \sum_{l \in L_i^{from}} X_{f_j^{in} l} = 0; \\ \sum_{l \in L_i^{to}} X_{f_j^{out} l} = 0; \sum_{l \in L_i^{from}} X_{f_j^{out} l} = 1. \end{gathered} \tag{7}$$

- A file f can be transferred from/to each site c_i at most once.

$$\forall f, i : \quad \sum_{l \in L_i^{to}} X_{fl} \leq 1; \sum_{l \in L_i^{from}} X_{fl} \leq 1. \tag{8}$$

- Each job j is executed at exactly one site c_i (site 8 in Figure 5):

$$\forall j : \sum_{i \in Sites} Y_{ji} = 1 \tag{9}$$

In addition, we use constraints for loop elimination, similarly, as it is described in [201]. The main principle of such constraints is that for each subset of r nodes C_r the number of transfers on internal links should be less than r. Excessive constraints can slow down the search procedure. In our case using such rule for $r = 2$ only has shown the best performance. Larger cycles can be eliminated after the schedule is created.

6.1.4 Constraints at the scheduling stage

At the scheduling stage the problem is to assign a start time for each task and a duration for file placement (S_j^J, S_{fl}^T, S_{fi}^F and p_{fi}^F). Here we assume the values of X_{fl} and Y_{ji} are fixed after the previous stage. At this stage we apply constraints on the duration and order of tasks as well as cumulative

constraints on the resources. The optimization criterion is the makespan, which is the latest completion time of all tasks. Since we already know that the latest task would be a transfer of an output file, we can define the makespan as follows:

$$C_{max} = \max_{l,j,f=f_j^{out}|X_{fl}=1} (S_{fl}^T + size_f \cdot o_l) \tag{10}$$

Figure 6 illustrates a schedule corresponding to the transfer path pre-

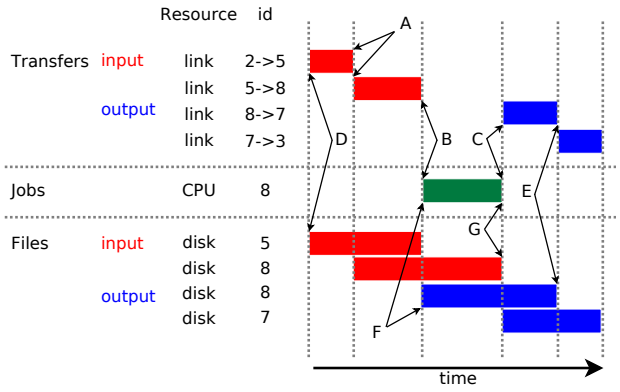


Figure 6: Example of a schedule for one job including related data transfers and placements.

sented in Figure 5. Each rectangle represents an active task at a resource. Red and blue colors indicate the input and the output files, respectively. Green color indicates job execution. Small black arrows and capital letters demonstrate examples related to the constraints described further. Please, note, that only a single example per constraint is referred, in order to keep the illustration tractable. In fact, the constraints consider many combinations of tasks, therefore, more examples can be spotted at the figure. Also, the input file placement at the source site 2 and the output file placement at the destination site 3 are not considered as a part of the problem, because those are permanent copies which are managed independently from our approach.

The following **constraints on the order and duration** of tasks are used at the scheduling stage:

- An outgoing transfer of a file from a site can start only after an incoming transfer to that site is finished (example A in Figure 6). The first

transfer of an input file from its source and the first transfer of an output file from the processing site are exceptions from this constraint.

$$\forall f, i, l_{from} \in L_i^{from}, l_{to} \in L_i^{to} | X_{fl_{from}} = 1,\ X_{fl_{to}} = 1 : \\ S^T_{fl_{from}} \geq S^T_{fl_{to}} + size_f \cdot o_{l_{to}} \tag{11}$$

- A job can start only after the input file is transferred to the selected processing site (example B in Figure 6):

$$\forall j, l, f = f_j^{in} : \quad S_j^J \geq S^T_{fl} + size_f \cdot o_l. \tag{12}$$

- An output file can be transferred only after the job is finished (example C in Figure 6):

$$\forall j, l, f = f_j^{out} : \quad S_j^J + p_j \leq S^T_{fl}. \tag{13}$$

- In order to guarantee enough storage space to complete each transfer, we assume that a file appears at a site at the moment when its incoming transfer starts (example D in Figure 6):

$$\forall f, i, l \in L_i^{to}, X_{fl} = 1 : \quad S^F_{fi} = S^T_{fl}. \tag{14}$$

- A file can be deleted from the start node of a link after the transfer is finished (example E in Figure 6):

$$\forall f, i, l \in L_i^{from}, X_{fl} = 1 : \quad S^F_{fi} + p^F_{fi} = S^T_{fl} + size_f \cdot o_l, \tag{15}$$

- In order to guarantee enough storage space for each new output file, we assume that an output file appears at a site at the moment when its job starts (example F in Figure 6):

$$\forall j, f = f_j^{out}, Y_{ji} = 1 : \quad S^F_{fi} = S^J_j. \tag{16}$$

- An input file can be deleted from the processing site after the job is finished (example G in Figure 6):

$$\forall j, f = f_j^{in}, Y_{ji} = 1 : \quad S^F_{fi} + p^F_{fi} = S^J_j + p_j \tag{17}$$

Cumulative constraints are widely used in CP to model resource usage by tasks. Each cumulative constraint requires that a set of tasks given by *start times*, *durations* and *resource usage* do not exceed a *resource limit* at any moment of time [200]. In our case we use three sets of cumulative constraints: for CPUs, storages and links. These constraints are summarized in Table 2.

Table 2: Summary of cumulative constrains on resources used in CP model.

Task	Resource	Start	Duration	Usage	Limit
Job j	CPUs at c_i	S_j^J	p_j	1	$NCPU_i$
Transfer of file f	Link l	S_{fl}^T	$size_f \cdot o_l$	1	1
Placement of file f	Storage at c_i	S_{fi}^F	p_{fi}^F	$size_f$	$Disk_i$

In order to define the cumulative constraints formally, let us consider arbitrary moment of time $t \in [0, \Delta T]$, where ΔT is the duration of a scheduling time interval. We define the three sets of cumulative constraints for our problem in the following way:

- CPU usage at each site c_i:

$$\forall t, i : \sum_{j|Y_{ji}=1, S_j^J \leq t \leq S_j^J + p_j} 1 \leq NCPU_i \tag{18}$$

  We sum over all the jobs (each uses one CPU) assigned to c_i ($Y_{ji} = 1$) which are starting before t and finishing after.

- Usage of each network link l (one transfer at a time can be active):

$$\forall t, l : \sum_{f|X_{fl}=1, S_{fl}^T \leq t \leq S_{fl}^T + size_f \cdot o_l} 1 \leq 1 \tag{19}$$

  We sum over all the file transfers assigned to the link l ($X_{fl} = 1$) which are starting before t and finishing after.

- Storage usage at each site c_i:

$$\forall t, i : \sum_{f|S_{fi}^F < t \leq S_{fi}^F + p_{fi}^F} size_f \leq Disk_i \tag{20}$$

Here we sum over all the files which are placed at the storage at c_i before t and removed after. If the file is never transferred to the site, both its start time and duration equal zero ($S^F_{fi} = 0$ and $p^F_{fi} = 0$) and it is not included in the sum. We do not consider initial sources and final destinations of the files, since their storage is managed separately from our scheduling.

6.2 Simulation, results

The proposed constraint satisfaction problem was implemented using MiniZinc [202] and Gecode [203] was used as a solver. The simulations were running under Windows 8 64-bit on a computer with Intel i5 (4 cores) 2.50 GHz processor and 6 GB memory. The Gecode solver was running in a parallel mode using 4 threads.

Four scheduling approaches were compared in the simulations:

Local: All the jobs are executed at the local site only. This strategy was used as a base line for comparison against other strategies.

Equal CPU load: Jobs are distributed between sites with the goal to maintain an equal ratio of job duration per CPU. Each job requests an input file transfer after it is started and waits for the output transfer to be finished. In many distributed computing models the input data are divided between computing sites based on static rules (fixed amount per site, per CPU or certain percentage of dataset). Since the job duration is typically proportional to the input size (within the same type of jobs), such approach will result in a similar load distribution.

Data transferred by job: Each CPU pulls a job from the queue when it is idle. Then it has to wait for an input transfer before the job starts and for an output transfer after the job is finished.

Optimized: This strategy is based on the model proposed in this chapter.

After the testing simulations, the first implementation of the scheduler was found to be excessively complex. A problem of realistic size featuring multiple sites and thousands of jobs required many hours to compute an acceptable schedule. Data production jobs typically last for approximately

two days. Therefore, the scheduling can be performed rarely allowing for the long solving time. However, in order to make the approach scalable to modern large-scale systems (featuring many tens of sites and hundreds of thousands jobs), a further simplification and optimization of the scheduler is clearly needed. For this reason, here we present the simplified simulations for the first implementation of the scheduler and discuss its limitations and further evolution in the next section.

We have limited our initial simulations to a configuration with three sites only and omitted constraints on storage. This allowed to set the time limit to 3 minutes for both planning and scheduling stages. The main purpose of these simulations was to check the potential of joint scheduling of jobs and transfers for optimization of data production. It also allowed to gain experience and ideas for further reconsideration of the scheduling approach.

The simulated environment consists of 3 sites: a central storage (with no CPUs) which is the single source for input files and the single destination for output files, a local processing site and a remote processing site. The slowdown of links between the central storage and the local site was set to 0, which means that the transfer overhead to/from the local site is negligible comparing to the remote site.

The slowdown of the links to/from the remote site was increasing in each simulation proportionally to a slowdown factor. The parameters of jobs were taken from log system of the STAR experiment. The considered jobs correspond to data production performed at computational site KISTI (South Korea) [204]. The average job duration was 2,760 minutes (46 hours) and average time of transfer was 5 and 10 minutes to/from the remote site respectively. We use the original values of the transfer times in the simulations where the slowdown factor equals one. Then, in further simulations, the transfer times increase proportionally to the slowdown factor. This allows to study the influence of network performance on job scheduling. In the simulated infrastructure 80 % of CPUs were available at the local site and 20 % at the remote site. 2,000 of jobs were scheduled stepwise by subsets (chunks) of 200.

The plot in Figure 7 shows the makespan improvement by different scheduling approaches compared to the job execution at the local site only. It is

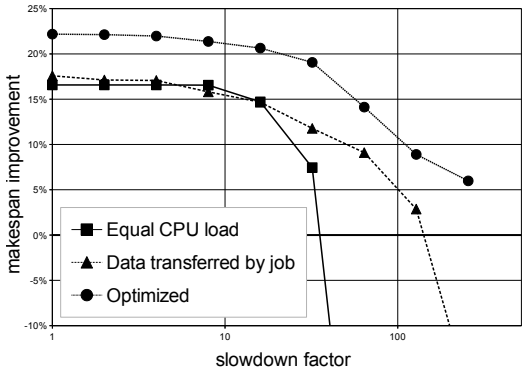


Figure 7: Makespan improvement of CSP based approach in simulations with real data.

calculated using the following formula:

$$makespan\ improvement_x = \frac{C_{max}^{local} - C_{max}^{x}}{C_{max}^{local}} \cdot 100\,\% \tag{21}$$

The curves shows the dependence of the improvement on the transfer overhead to the remote site which increases proportionally to the slowdown factor. When the transfer overhead becomes significant both heuristics ("Equal CPU load" and "Data transferred by job") fail to provide an efficient usage of the remote resources (the makespan improvement goes below zero). Negative makespan improvement means that, in this case, it would be faster to process all the data locally than to send its part to a remote site relying on the heuristic. The proposed global planning approach (Optimized) systematically provides a smaller makespan and adapts to the increase of transfer overheads better than the other simulated heuristics. It is able to provide a positive gain in makespan by using remote resources even when the transfer overhead is comparable to a job duration.

6.3 Limitations of the model

A problem of scheduling of distributed data production was formulated in a form of a constraint satisfaction problem where CPU allocation, storage usage and data transfer were mutually considered. This model provides a formal description of the general problem we address in this thesis. The simulations of simple cases of distributed data production, where our first approach was

compared against other general policies, have shown that scheduling of CPU allocation coupled with data transfer has a potential to improve an overall processing throughput. The advantage of such approach is more significant in cases when network performance becomes the bottleneck.

The simulations based on data extracted from log systems of the STAR experiment have shown that the proposed global planning approach systematically provides a smaller makespan and adapts to the increase of transfer overheads better than the other simulated policies.

Despite the demonstrated potential of the described approach, the study has revealed significant limitations. The first important drawback is the computational complexity of the underlying CSP. The model has shown a reasonable solving time in the presented use case, however, its scalability to real large-scale systems appeared troublesome.

The excessive complexity of this CSP formulation is due to inclusion of all possible permutations of job order and assignment to resources into the search space. While such permutations are important for general workloads, they are irrelevant for data production and similar processing types. Therefore, a scheduling approach for distributed data production can greatly benefit from further simplification. Also, the CP formulation of cumulative constraints introduces additional complexity. Effectively, it produces a separate expression for each pair of site and possible time value. It would be preferable to describe resource usage with a smaller number of expressions.

The second drawback of the model is uncertainty of some of its inputs. In real applications only the input size of a job is known in advance. The job duration and size of its output can only be estimated. Possible error of such estimation would often lead to the violation of the schedule and repetitive rescheduling.

The third drawback is also related to the precision of the schedule execution. In a dynamic large-scale environment where performance of all resources can fluctuate it would be difficult to fulfill the schedule where the start/end time of each operation are precisely set. Either a scheduler performance should allow to redo the scheduling periodically and efficiently or the schedule and its execution should account for certain deviations.

Last, but not least, the makespan estimation at the planning stage (Equations 1, 2 and 3) does not consider overlap of tasks (e.g. jobs with input

transfers and output transfers) and their time interdependence. Such consideration is needed in order to achieve a predictable load balance. However, in this initial model, it can only be achieved after the scheduling stage. The analysis of possibilities to further improve load balance before instantiating the complete schedule has inspired our next model.

Using the lesson from this study we have developed a new job scheduling approach which we present and validate in Sections 7 and 8, respectively. That approach addresses the drawbacks of the first model. It is based on the network flow maximization problem which provides a better complexity compared to CSP formulation. The approach is dedicated specifically to data production (and similar applications) and utilizes its properties to simplify schedule generation and execution. In particular, it does not require to compute exact start time for each job and transfer, dynamically adapts to monitoring data and is not vulnerable by deviations during execution. It distributes the load among the resources in such a way, that maximizes processing throughput while neither exceeding available capacity nor compromising makespan.

7 Planning of distributed data production

Using our previous experience in data production scheduling and observations in the case studies we developed a new approach to the problem. In order to achieve a reasonable complexity of the scheduling problem, we used specific properties of data production. First, in a single campaign, each portion of data has to be processed exactly once. Second, all the computational jobs are independent and interchangeable which means that they can be executed in arbitrary order. Moreover, since a single data production campaign consists of a large set of similar files undergoing the same type of processing, the parameters of upcoming jobs can be predicted using the statistics of previously finished ones. Despite particular jobs can vary in their parameters, for a large enough dataset and long enough time interval we can rely on the average values. Following these assumptions, the main idea of our approach is to plan resource load in advance for a limited time step (planning time interval) and then distribute data and computational jobs accordingly, rather than producing a complete schedule. Planning for limited time intervals (e.g. 12 hours) provides adaptability necessary for dynamic systems. Planning repeatedly for shorter time intervals allows to deal with uncertainties by correcting predictions and adjusting to the current state and performance of the resources.

Figure 8 illustrates the input for our planner: at the beginning of each

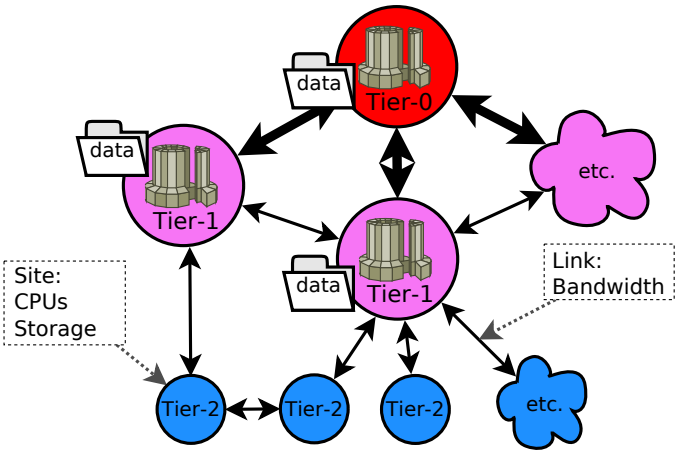


Figure 8: Input for the network-flow based planner: distributed data production problem represented as a graph.

planning time interval it considers current data location, state and load of the resources, structure and bandwidth of the network. The planner defines how much data (input and output) should be transferred over each link within given time interval. The plan is produced with the goal to maximize the number of used CPUs but avoid network congestion or running out of storage space. There is a local queue of input files maintained at each computational site which is kept long enough (when possible) to saturate its CPUs with jobs without the need to access data from external storage. The planner uses the network flow maximization algorithm [153] with polynomial time complexity which is a significant improvement compared to general scheduling algorithms [157, 153].

The plan is executed by dedicated services running at each computational site called "handlers". Each handler is responsible for data at the storage of its site. It submits data for local processing or forwards it to neighboring sites with respect to planned data flows. The output data are also transferred by the handlers to its destination in accordance with the plan.

7.1 Elements of the model

The raw data from the detector are being automatically archived to the *central storage* at the main computational facility of the experiment c_0 (called Tier-0 site). The data are kept in the form of separate files (typically of several gigabytes of size) which can be copied to other destinations for processing. We refer those files as *input files*. The outcome of the processing is called reconstructed data, and also consists of separate files (also several gigabytes of size) which we refer as *output files*. All the output files are being persistently stored to the central storage as well. Large collaborations, such as CERN experiments, typically use multiple sites for long term data storage in addition to the Tier-0 site. Such sites are referred as Tier-1s. Significant portions of data are replicated from the Tier-0 to Tier-1s in order to improve data access and safety against loss. The actual distribution of data between such sites depends on data management model, history of computations and plans of the collaboration.

In our problem a *(computational) site* $c_i \in C$ is a set of closely connected machines (can be also referred as a data center, computer cluster, computing facility, farm or a server) which has a fast access to a common data storage

(referred as a *local disk* of size $Disk_i$) and a shared connection to the outer network. A computational facility of a scientific institution is an example. The key principle to identify a set of machines as a computational site is that they can access a given local disk with a latency which is negligible compared to their access to other storages (like the central storage) in the distributed system.

We assume that there is a common job scheduling system (*local scheduler*) which allows to submit jobs to all of the machines combined into a site. Our approach is not intended to replace general purpose CPU schedulers at sites. From the point of view of a local scheduler, there are many distinct users concurrently submitting their jobs (and data production can be seen as one of such users). The local scheduler allocates CPUs to jobs according to its internal policy. For example, a quota may be set to limit the number of CPUs reserved for data production specifically. We monitor CPUs (slots) provided for data production at each site, e.g. using the number of active pilot jobs [205], job agents [206], virtual machines [207] or containers [4]; and supply them with data for computation. Their number can change in time depending on concurrent workload, availability of resources, usage quotas and other factors. For simplicity, in the rest of the chapter we refer a CPU number $NCPU_i$ at a site i, as if each data production job uses a single CPU and there are no other concurrent jobs in the system. $NCPU_i$ is updated over time according to monitoring data. Our goal is to maintain a stable queue of input files prestaged at the local disk of each site so that new computational jobs do not wait for an external data transfer. The input files from the local queue can be submitted for processing or transferred to another site if needed. At the same time, we need to plan transfers of output data to its destination. The main idea of our optimization is to ensure that during data production the transfers are performed in such a manner that neither disk quota nor network capacity are exceeded while the input queues at sites are kept at a reasonable size.

There are three functions that a site can serve during data production: input source, computational site and output destination. Those functions are not exclusive, which means that a particular site can be enabled with all three, two or just one function. Also, there can be many sites with the same function in the computational network. *Input source* is a site which already

has a portion of input data at its local disk by the time when data production starts. A *processing site* is the one which can process jobs ($NCPU_i > 0$). *Output destination* is a site selected to store the outcome of data production on its persistent storage. As soon as an output file is transferred to any of possible output destinations we assume that its processing is finished. For each site c_i we consider the total size of currently available input files k_i and the currently available free space to store new output files $\overline{k}_i$. Those values are set to zero if the site is not assigned the corresponding function.

The sites are interconnected with network links $l \in L$ of a given bandwidth b_l so that they form a grid, which can be represented by a weighted graph. We consider bandwidth as an approximation of an observed data transfer rate, which can be estimated as an amount of transferred data per unit of time. In our approach the graph is not necessarily fully connected. It should be understood as a high-level (but realistic) representation of a detailed network structure. For instance, if multiple end-to-end connections between sites share a part of a network path, such part should be modeled as a single link rather than independent links for each connection. For a better detailed modeling, network routers can be represented by nodes with zero number of CPUs and none local disc size.

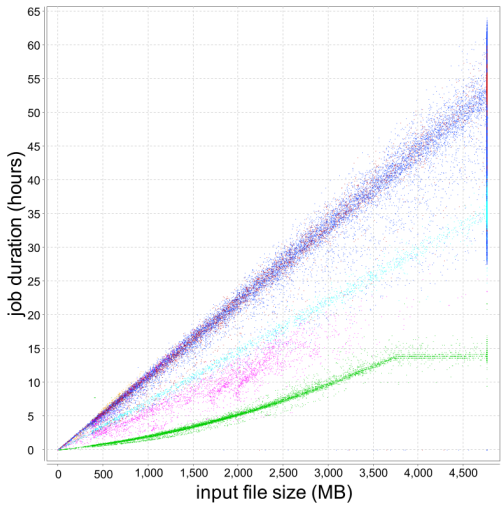
There is no requirement on the order of particular jobs. Each computational job j processes a unique input file of size $size_j^{in}$, produces a unique output file of size $size_j^{out}$ and has a duration p_j. Only the size of the input file is known in advance, before the job is finished. We can estimate two other parameters by:

$$p_j \approx \alpha_i \cdot size_j^{in} \tag{22}$$

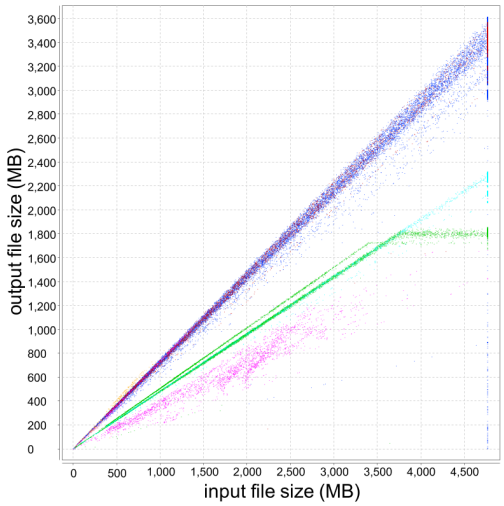
and

$$size_j^{out} \approx \beta \cdot size_j^{in}, \tag{23}$$

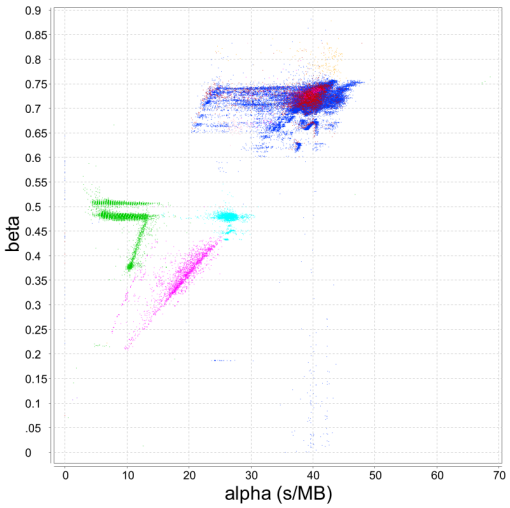
where α_i is an average processing speed of a single CPU at a site i and β is an output-to-input size ratio. These two values are considered to be constant coefficients. Since all the jobs perform the similar type of processing, we can use average values of α_i and β of previously completed jobs. Figure 9 shows the relation between duration, input and output size of a sample of 60,000 data production jobs of the STAR experiment. Colors represent different types of jobs. At Figure 9a one can observe that the job duration depends linearly on the input size in most of the cases. Similarly, Figure 9b shows



(a) Dependence of job duration on input size.



(b) Dependence of output size on input size.



(c) Values of α and β.

Figure 9: Relation between duration, input size and output size of data production jobs of the STAR experiment.

the linear dependence of output size on input size. Parameter β is less than 1 for all the jobs in the analyzed sample. Finally, it is visible in Figure 9c that values of α and β are close (clustered at limited regions of the plot) for each type of data production jobs.

7.2 Planner based on network flows

Let us consider a scheduling time interval ΔT at an arbitrary moment of data production. We assume that at the starting moment of ΔT some of the CPUs in the grid are already running production jobs, and there can be some amount of input data already placed at each site. We need to transfer the next portion of data to each site during ΔT in order to avoid draining of the local queues by the end of this interval. We will give two separate problem formulations: for an input and an output transfer planning based on network flows.

In order to formulate a network flow maximization problem [153] for file transferring we define a capacitated $\{s, t\}$ network, which is a set of vertexes V including a source s and a sink t; and a set of edges $e \in E$ with their capacities cap_e, representing the amount of data which can be transferred during ΔT. A solution that assigns a non-negative integer number $flow_e$ to each edge $e \in E$ can be found in polynomial time with known algorithms [153]. Here $flow_e$ is understood as an actual amount of data to be transferred (or processed) within ΔT. In the further text, we use the term *edge* to denote a general element of a network flow problem (including dummy edges). In contrast, by *link* we refer such edge which corresponds to a real network connection.

We prioritize transfer of output files because it allows to create sufficient free space for upcoming input files. For this reason, we solve the output problem first, and then use its solution to calculate the remaining capacities of the network links.

7.2.1 Output flow planning

In order to transform a given graph of a grid into a capacitated $\{s, t\}$ network for the output transfer problem we add two dummy vertexes: a source s and a sink t and dummy edges. The source s is connected to each processing site c_i with a dummy edge $\overline{d}_i \in \overline{D}$. Its capacity $\overline{w}_i$ defines the maximum expected amount of the output data to be transferred from the site. Its calculation is explained in Section 7.2.3. Each output destination site c_i is connected to the sink t with a dummy edge $\overline{q}_i \in \overline{Q}$ having capacity $\overline{k}_i$ – the currently available free space to store new output files. In this formulation capacity

of each edge defines the maximal amount of data that can be transferred within time interval ΔT. For each real network link $l \in L$ with bandwidth b_l it is $b_l \cdot \Delta T$. The transformation is illustrated in Figure 10, where solid lines

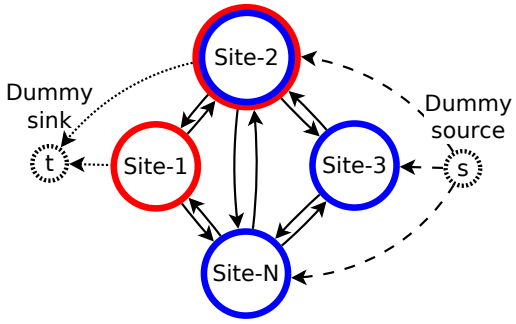


Figure 10: Capacitated $\{t, s\}$ network for the *output* planning problem.

are network links L, dotted lines are dummy edges $\overline{Q}$, dash lines are dummy edges $\overline{D}$. Output destinations are in red cycles, processing sites are in blue cycles (Site-2 shares both functions).

The following expression summarizes the capacities of edges in the output problem:

$$cap_e = \begin{cases} b_e \cdot \Delta T & \text{if } e \in L \\ \overline{w}_i & \text{if } e = \overline{d}_i \in \overline{D} \\ \overline{k}_i & \text{if } e = \overline{q}_i \in \overline{Q} \end{cases} \quad (24)$$

We denote the solution for the output transfer problem as $flow_e^{out}$. It specifies the amount of output data that has to be transferred over each link during ΔT.

7.2.2 Input flow planning

For the input problem, we apply a similar transformation to the initial graph of the computational network as for the output problem. We add dummy edges $d_i \in D$ from each processing site to the sink, and dummy edges $q_i \in Q$ from the dummy source s to input sources. These dummy edges allow us to introduce constraints on the storage capacity, CPU throughput and data availability at sites. For real network links $l \in L$ we also take into account the capacity reserved for output transfers $flow_e^{out}$. The network for the input problem is presented in Figure 11, where solid lines are network links L,

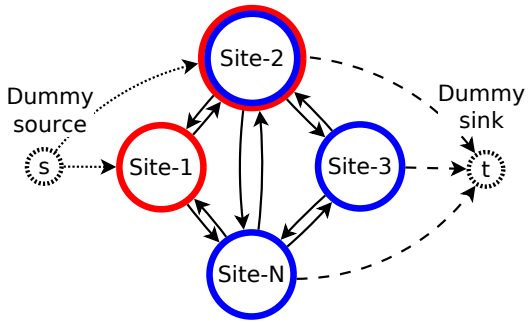


Figure 11: Capacitated $\{s, t\}$ network for the *input* planning problem.

dotted lines are dummy edges Q, dash lines are dummy edges D. Input sources are in red cycles, processing sites are in blue cycles (Site-2 shares both functions). To summarize, the capacities of edges in the input problem are defined as follows:

$$cap_e = \begin{cases} b_e \cdot \Delta T - flow_e^{out} & \text{if } e \in L \\ w_i & \text{if } e = d_i \in D \\ k_i & \text{if } e = q_i \in Q \end{cases} \tag{25}$$

where w_i is the demand for new input data at processing site c_i. Its calculation is explained in the Section 7.2.3. We denote the solution for the input transfer problem as $flow_e^{in}$. It specifies the amount of input data that should be transferred over each link l during ΔT.

7.2.3 Capacities of dummy edges

Here we explain how the capacities of dummy edges w_i and $\overline{w}_i$ are calculated in order to complete the maximum flow problem. These capacities define the expected net amount of input/output data to be transferred to/from each site during the planning time interval ΔT.

Let us start with a simple case. We call it a *steady state* when during the entire previous planning cycle the site has kept its peak performance (all the available CPUs are processing data) and no changes are made to its configuration. In this case we can assume that the site will process/produce data at the same rate during the next ΔT. Therefore, we can set w_i to be equal to the amount of input data processed and $\overline{w}_i$ to the amount of output data produced at this site during the previous planning cycle.

In other situations when we cannot rely on previous statistics, we need an estimation based on the current state of the site. Examples of such situations are: the very first planning cycle, changes to a site's configuration or shortage of free disk space. Also, it is desirable to increase dataflow to a site with underutilized CPU capacity when possible.

Consider transfer of output files first. Let I_i^{out} be the initial size of output data (of previously finished jobs) which are ready to be transferred from a local storage. We also need to estimate the amount of new output data of jobs that will finish during ΔT. If most of the CPUs at the site are saturated, than this value can be estimated as $\frac{\beta}{\alpha_i} \cdot NCPU_i \cdot \Delta T$. In the opposite case, if not all the CPUs are busy, such estimation may become exaggerated. Since ΔT is assumed to be smaller than the average job duration it is unlikely that a job started within ΔT will also finish within it. Therefore, the considered value cannot be greater than the total size of output files of currently running jobs C_i^{out}. To find the bottleneck value, we have to select the smallest estimation. The final formula of the capacity of dummy edges for output problem is:

$$\overline{w}_i \;=\; I_i^{out} + \min(\frac{\beta}{\alpha_i} \cdot NCPU_i \cdot \Delta T, \;\; C_i^{out}) \tag{26}$$

The amount of input data which can be transferred to the processing site is limited by two factors: size of the local disk and the CPU throughput. The incoming data flow should adjust to the processing throughput in order to keep the input queue at a constant size. If the queue drains some jobs get delayed. On the other hand, if the queue grows too large, it would be more efficient to send files to a less loaded site.

Let us denote the available free disk space at the site i as R_i. In order to avoid running out of free space at the local disk, we set an upper limit δ for a planned disk usage. This ensures that there is always enough space for an output file of a new job or for an incoming file. This is especially important, since job durations and file sizes can fluctuate from the average values which are used for the plan generation. In other words, we assume that $Disk_i \cdot (1 - \delta)$ of disk space is planned to remain free.

If there are free CPUs at the site, each incoming input file will be immediately submitted for processing, creating a reservation for a new output file of size $size_j^{out} = \beta \cdot size_j^{in}$. For this reason, the constraint on the storage size

gives us $\frac{R_i-(1-\delta)\cdot Disk_i}{1+\beta}$. On the other hand, in order to saturate unused CPUs, we need to transfer at least one file for each of them. If an average size of an input file is $size^{in}_{avg}$ and the total size of input files in local queue is I^{in}_i, than the required amount of input data is $NCPU_i \cdot size^{in}_{avg} - I^{in}_i$. We should select the minimal value between the storage and CPU constraints, in order to find the bottleneck. The final expression for the capacity of the dummy edges for the input problem is:

$$w_i = \min(\frac{R_i - (1-\delta)\cdot Disk_i}{1+\beta}, NCPU_i \cdot size^{in}_{avg} - I^{in}_i) \tag{27}$$

In the Eqn. 26 and 27, ΔT and δ are parameters of the scheduler. All the other used values are obtained from monitoring data right before each planning iteration.

7.2.4 Solving Procedure

We perform planning by *scheduling cycles*: instantiate a plan for a fixed time interval ΔT and repeat until all the data are processed and the output is transferred to the central storage. Each plan is created at the beginning of its interval using updated monitoring data. Therefore, each plan relies on the current system state but not on previously issued plans.

The problems of input and output transfer over the same network can be solved independently if input and output flows do not compete for bandwidth. A competition may occur if input and output data are transferred in the same direction over the same link and there is not enough bandwidth to accommodate both flows. In a steady state such competition is unlikely, which can be proven under assumptions: (a) all the real network links are full-duplex, i.e., transmissions in both directions are independent and have the same bandwidth (b) in a steady state the size of the output transferred from each site is proportional to the size of the input transferred to that site in each scheduling interval, i.e., $flow^{out}_{d_i} \approx \beta \cdot flow^{in}_{d_i}$, where $\beta \leq 1$.

Let us consider two distinct computational sites c_1 and c_2 connected by two opposite directed links $l_1 = (c_1, c_2)$ and $l_2 = (c_2, c_1)$ with equal capacities $cap_{l_1} = cap_{l_2}$. If a solution of the input transfer problem assigns flows to this links such that $flow^{in}_{l_1} \geq flow^{in}_{l_2} > 0$ then we can substitute such solution $flow^{in}_e$ with a new one $\widehat{flow^{in}_e}$ where $\widehat{flow^{in}_{l_1}} = flow^{in}_{l_1} - flow^{in}_{l_2}$, $\widehat{flow^{in}_{l_2}} = 0$

and flows over the rest of the links are unchanged. The same is true for output transfer. This proves that in an optimal solution the same type of files (input or output) are transferred between any two nodes in one direction only, i.e. over one of directed links only.

If we have a solution for the input flow maximization problem $flow_e^{in}$ we can produce a solution for the output problem $flow_e^{out}$ such that for any opposite pare of links $l_1 = (c_1, c_2)$ and $l_2 = (c_2, c_1)$ the output flow is $flow_{l_1}^{out} = \beta \cdot flow_{l_2}^{in}$ and since $\beta < 1$ the capacity of links is not exceeded $flow_{l_1}^{out} = \beta \cdot flow_{l_2}^{in} \leq flow_{l_2}^{in} \leq cap_{l_2} = cap_{l_1}$. Due to symmetry of the two problems, this solution $flow_e^{out}$ is also the maximum flow for the output transfer problem. Combining this with what was proven in the previous paragraph, if $flow_{l_1}^{in} > 0$ than $flow_{l_2}^{in} = 0$ and thus $flow_{l_1}^{out} = 0$. This means that in a steady state input and output files are never transferred over the same link. And thus, maximum flow problems for input and output transfers can be solved independently.

In a reality, the system will not always operate in a steady state due to fluctuations from average parameters and addition / withdrawal of the resources. In order to resolve possible concurrency we plan the output transfer first, and then use remaining network capacity to plan the input transfer. We prioritize the output transfer because it allows to free space for upcoming input data. The solving procedure for a single scheduling cycle consists of the following steps:

1. Calculate the amount of output data $\overline{w}_i$ to be transferred from each site as defined by Equation 26.

2. Construct the network of the output problem as described in Section 7.2.1.

3. Solve the network flow maximization problem for the output network. Obtain output data flows over real links $flow_e^{out}$ from the solution.

4. Calculate the demand for input data w_i at each site as defined by Equation 27.

5. Construct the network of the input problem as described in Section 7.2.2.

6. Solve the network flow maximization problem for the input network. Obtain input data flows over real links $flow_e^{in}$ from the solution.

The resulting input $flow_e^{in}$ and output $flow_e^{out}$ flows over the real network links present the plan of data transfer and computation for the next time interval ΔT. Because the capacities of the dummy edges are adjusted to the current situation, this approach can be directly used from the start of the data production to its end.

7.3 Plan execution

After a plan for a time interval ΔT is created it has to be executed at sites. We assume that there is a dedicated service running at each site which is responsible for sending statistics to the planner, receiving the plan and executing it. We call such a service a *handler*.

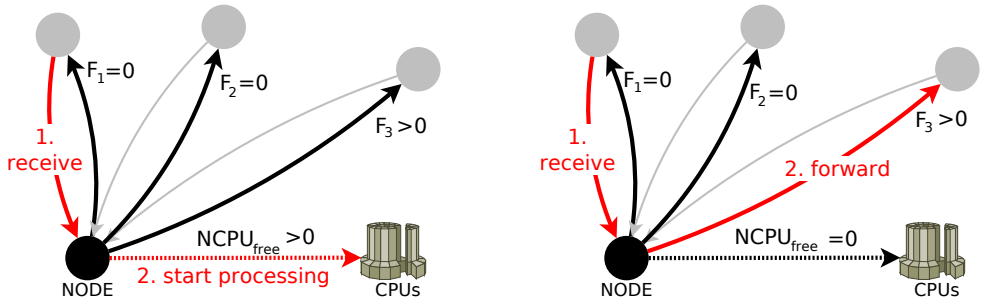
The handler is responsible for transfers over outgoing links of its site. It keeps counters of how much data of each type (input and output) remains to be sent from its site to the neighboring sites during the current scheduling cycle. This implies two counters F_l and $\overline{F_l}$ (input and output) for each outgoing link l connected to the site. When the handler receives a new plan it updates the counters to be equal to the flows over the corresponding edges. If the handler fails to fulfill the plan precisely, the system automatically recovers from such an error, since each scheduling cycle relies on the current state of the system and performance statistics, but not on the execution of previously issued plans.

During the plan execution, each time when a new file arrives at the site, the handler decides either to keep the file for local processing or to forward it to another site. In order to make the decision, the handler goes through the following list of options and executes the first appropriate one, depending on whether its requirements are satisfied:

1. If the received file is of input type and if there is a free CPU at this site then the file is submitted for processing (see Figure 12a).

2. Otherwise, if there is a link with a counter which is greater than zero for the corresponding file type (input or output) then the file is sent over that link (see Figure 12b). The counter is decreased by the size of the file.

3. Otherwise, the file is kept at the local storage until it can be processed (when a CPU becomes free or a new plan arrives).



(a) Handler submits a received input file for processing.

(b) Handler forwards a received input file to another site.

Figure 12: Plan execution by handlers at sites.

Such an order of options ensures that the file is processed as soon as it arrives to the site with a free CPU, and all the CPUs are busy as long as there are unprocessed input files at the local disk. No excessive transfers can occur in the system because a solution to the maximum flow problem contains no cycles and an input file is not forwarded unless all the CPUs are busy.

Another important role of the handler is to check the consistency of each newly received file and confirm it to the sender. Only after the confirmation, the transferred file can be deleted from the sender site, otherwise, data loss may occur.

When processing of an input file starts, the handler makes a reservation for the output file at the local disk, and when the processing is finished the handler deletes the input file.

7.4 Balance between multiple data sources

The maximum flow problem can have multiple solutions, which means that in certain networks the maximum total flow can be achieved by several alternative flow assignments to edges. Intuitively, for a large enough network, when the total flow is limited by the capacity of a subset of the edges (the bottleneck) the flow over the remaining part of the network can be routed in many different ways. This fact raises an important issue when it comes

to planning with multiple input sources. An example is given in Figure 13. S1 and S2 are input sources, P1 – P3 are processing sites. The dash and dot-

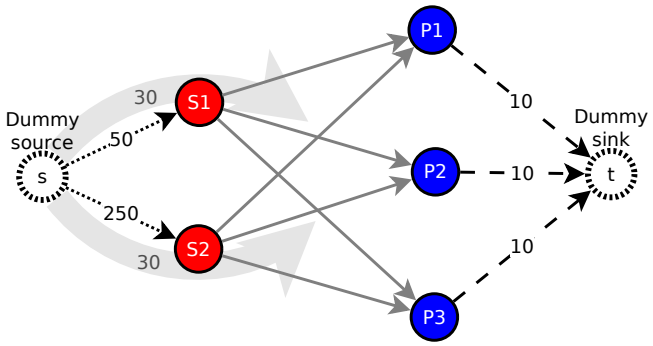


Figure 13: Multiple solutions of the maximum flow problem.

ted lines are dummy edges, labels show their capacity, solid lines are network links. The gray arrows in the background show two alternative flows through the input sources. Here the total flow is limited by CPU throughput, while the network structure allows to select from which input source to transfer the data. However, classical network flow maximization algorithms do not select between solutions with the maximal flow on any additional criteria [153]. When an input source gets depleted, the number of possible transfer paths decreases and it potentially leads to the emergence of additional bottlenecks. For this reason, a solution taking into account the remaining amounts of data at the sources is needed to properly balance their usage. A helpful strategy is to utilize sources with more data as much as possible from the beginning but try to keep the smaller sources for later and utilize them when it allows to maximize the overall flow.

With this concern, additional criteria for selection between multiple maximum flow solutions should be added to the problem. We have achieved this by extension from the maximum flow problem to the *minimum cost maximum flow* problem in our planner. For such transition, we assign a cost $cost_e$ for each edge $e \in E$ in addition to previous problem formulation. The cost of a flow function $flow_e$ for a given graph is defined as $\sum_{e \in E} flow_e \cdot cost_e$. A *minimum cost maximum flow* (min-cost max-flow) of a given network is a maximum flow with a smallest possible cost. Known algorithms such as gen-

eralized push-relabel algorithm [208] can solve min-cost max-flow problem in polynomial time.

In order to balance usage of multiple input sources, we assign costs to dummy edges q_i depending on the amount of input data remaining at the sources. At the beginning of each planning cycle our planner does the following:

1. Sort input sources in descending order by the amount of available input data.

2. Set costs of the dummy edges q_i depending on the rank of the source i in the sorted list. In the current implementation, the cost is set equal to the rank.

3. Set the cost of the rest of the edges to one in order to take distances into account.

Since the costs are updated at each planning cycle the priorities of the sources change as they are getting depleted.

Influence of the particular values of costs can be better understood with the following simplified example. Let us consider two sources with costs $cost_1$ and $cost_2$. If there is a computational site at a distance (length of the network path given that cost of real links is set to one) d_1 from the first source and d_2 from the second, the cost per unit of flow to that site would be $cost_1 + d_1$ and $cost_2 + d_2$, respectively. The site with a smaller resulting cost will be selected to transfer the data (if there is enough network capacity). For example, the first site will be selected if $cost_1 - cost_2 < d_2 - d_1$. This shows that the difference between costs assigned to the sources defines an additional number of transfers which are performed to establish the balance between sources. In order to avoid excessive transfers, the range of costs assigned to the sources should be kept reasonably small.

7.5 Initial data distribution

We can introduce an additional stage into our scheduling approach which will allow to improve data locality prior to computations. This stage is not mandatory and is designed to be executed in cases when storage and network resources are accessible before the CPUs are pledged for the data production.

For this reason, the time required for the initial data distribution is not included in the resulting makespan. This initial data distribution can be performed using our planning approach with several modifications which are described in the following text. A carefully planned initial data distribution can help to reduce the subsequent data production makespan. In the real world, it is usually possible to move data across several available storages before the computation starts. For example, the STAR collaboration makes agreements with external institutions to use their computational facilities for data production during a predefined time period [21], in this case, the access to the remote storage is usually granted before the access to CPU resources. Under such conditions, it would be advantageous to move a part of data to that remote storage before the actual computations. Another example comes from the ATLAS experiment [4]. There, the raw data are persistently stored at 12 geographically separated sites. Those raw data are (re)processed (typically several times during years) using updated algorithms and calibration data in order to improve quality of the resulting reconstructed data. Such processing passes are planned by the experiment's collaboration in advance, so that the dataset to be processed and computational resources to be used are known in advance. This gives an opportunity to adjust the data distribution in order to decrease the makespan of the upcoming data processing.

The general idea of the initial planning stage is to consider the entire data production in one planning cycle where ΔT equals to an estimated makespan. The produced plan is not expected to be highly accurate due to the huge planning time interval, but it allows to find how much data should be taken from each possible source. After the calculated data distribution is established the data production can start as described before.

7.5.1 Model description

Let Z be the total size of input data to be processed. Some of the sites can be used as input sources, the maximal amount of input data, which can be placed at such site is $K_i > 0$. The task is to find how much data k_i should be placed at each source and how long the data production will take.

We can again apply the min-cost max-flow approach to the initial data distribution problem. Similar as before (Sections 7.2.2 and 7.4), we transform

the graph of the computational network into capacitated $\{s, t\}$ network and set costs to all its edges. We set the capacity of each real network link equal to the amount of data which can be transferred over it during the data production $cap_e = b_e \cdot \Delta T$. The dummy source s is connected to each input source via dummy edges q_i with capacity K_i. Each processing site c_i is connected to a dummy sink t via a dummy link d_i with capacity $cap_e = \frac{NCPU_i}{\alpha_i} \cdot \Delta T$ which estimates its data processing throughput. The resulting network is the same as in Figure 11, yet the capacities of the edges are set as follows:

$$cap_e = \begin{cases} b_e \cdot \Delta T & \text{if } e \in L \\ \frac{NCPU_i}{\alpha_i} \cdot \Delta T & \text{if } e = d_i \in D \\ K_i & \text{if } e = q_i \in Q \end{cases} \tag{28}$$

The coefficient α_i should be derived from the statistics of previously finished jobs; its variance can be significant given the heterogeneity of resources and jobs. For this reason, the accuracy of α_i estimation is a limiting factor for the precision of the resulting initial plan. However, as explained before, the initial plan does not have to be strictly fulfilled during the upcoming data production itself.

In the current implementation, the costs of all of the edges are set to one. This allows to reduce the number of transfers during the computational stage, but potentially increases the number of required transfers before the computations. Alternatively, the costs of dummy edges q_i to the source sites which do not contain significant portions of data yet can be set to one, while the costs of the rest of the edges can be set to zero. As the result, the planner will try to utilize existing placements of data as much as possible (and will use additional sources only if this will allow to avoid network bottlenecks), however, the number of transfers during the computational stage may increase. The choice of alternatives should depend on the particular use case. The advantage of the proposed approach is certainly its flexibility which allows to adjust to real life conditions.

7.5.2 Solving procedure

The makespan of data production ΔT is not known in advance, in order to find it we start with an estimation and then improve its value iteratively. The iterations continue until we find a value of ΔT for which the maximum total

flow Φ equals to the total size of input data Z with a predefined precision ε. The overall solving procedure for the initial planning problem is the following:

1. Calculate an optimistic (as if there are no network bottlenecks) estimation for makespan $\Delta T = \frac{Z}{\sum \frac{NCPU_i}{\alpha_i}}$.
2. Construct the $\{s, t\}$ network as described in Section 7.5.1.
3. Update capacities of the edges using the current value of ΔT.
4. Solve the max-flow min-cost problem. If $\mid Z - \Phi \mid < Z \cdot \varepsilon$ then go to (6).
5. Set a new value for the estimated makespan $\Delta T = \frac{Z}{\Phi} \cdot \Delta T_{previous}$ and go to (3).
6. Resulting flows $flow_e$ over dummy edges q_i are the amount of data to be placed at each source ($k_i = flow_{q_i}$), ΔT is the expected makespan.

To summarize, this procedure allows to find a makespan estimation ΔT for data production and amount of the input data k_i to be initially placed at each source site i.

7.6 Data replication

The data flow of a HENP experiment begins at a site where the detector is located. Typically, the site (tier-0) also provides a persistent storage for all the data related to the experiment. In addition to the main storage, overlapping subsets of data are copied across multiple Tier-1 sites [4]. The degree of replication for particular files can vary and change in time depending on the computation history and collaboration plans. Each file can be assigned a logical filename (*lfn*) uniquely defining it. A single *lfn* can point to multiple physical files identified by physical filenames (*pfn*).

We assume that the data are replicated before the data production starts. The planner never deletes those initial (persistent) file replicas. It considers initial data locations and creates temporal file replicas when it allows to speed the data access up. Therefore the planner distinguishes two types of a physical file instance:

- *Persistent* instance is the one that was created before the data production started and has to be kept.

- *Temporal* instance is created when the file is transferred to a new site as a result of the plan execution.

Only one temporal instance per *lfn* can exist at a time. This instance is deleted after the file is processed. Forwarding a temporal replica from one site to another implies creation of a new temporal replica at the destination and deletion of an old one from the source. The temporal replica can be understood as a "traveling" instance which hops between sites according to the plan until it gets processed.

There is a central service (called file catalog) which keeps track of all the *lfn*'s of the input files in data production. The handlers at individual sites can communicate through this service in order to ensure that all *lfn*s are processed exactly once and data are never transferred to a site where another its copy is already present. File catalog provides information on the number of physical instances for each *lfn* and their location. It also stores the status information related to each *lfn*. Two statuses are used in the current implementation:

- *Queued* status means that the *lfn* waits to be processed, no temporal copy was created, neither was the job started.
- *Used* status means the file is being processed at one of the sites which has its persistent copies, or a temporal copy was created to be processed at another site.

Initially, all the file statuses are set to "queued". When a file is selected to be processed at one of its initial locations the corresponding *lfn* status is changed to "used". When a file is transferred from its initial location the corresponding *lfn* status is changed to "used" and a temporal replica is created at the destination. From this moment, only this temporal replica is allowed to be processed. Through the update of the status the other sites with a persistent replica of the file are prevented from processing or sending it out again. Since a max-flow problem solution contains no flow cycles the temporal instance travels a limited path in the grid until it is processed, for this reason, no excessive transfers take place.

At the beginning of data production each handler at a source site:

1. Registers persistent replicas at its site to the file catalog and sets their status to "queued".

2. Organizes input files at the site into a queue where the files with fewer replicas are prioritized. This allows to process the files with fewer replicas first in order to leave more options for later planning cycles.

During the data production, each handler periodically scans its local queue and removes files with status "used" from it. Incoming input files forwarded from other sites (the temporal instances) are stored and placed at the beginning of the queue. Whenever there is a capacity to process a file or to send it to another site, the handler takes the next file from the queue. If it is a temporal instance, the file can be used without check since only a single temporal replica can exist. Otherwise, if it is a persistent instance, the handler checks its status.

- If the status is "used" then discard it from the queue and proceed to the next one.
- If the status is "queued" then use the file and change its *lfn* status to "used".

Each new plan depends on the current state of the system but not on previously issued plans. Therefore, failures during execution of a plan do not affect future planning cycles. Recovery can be done using standard approaches. Loss of replicas at a given site would be detected by handlers and removed from the possible replicated candidates. File corruptions can be detected using checksums stored in the file catalog. The next cycle of planning would consider a new file distribution landscape. Timed-out and failed actions can be re-queued within one planning cycle. A handler can perform self-recovery at the start of a planning cycle (i.e. when a new plan is issued). In such case, a handler verifies the content of the local disc and running jobs and then it can proceed with the current plan. In case of a planner failure, it restarts and requests current status from all the sites and then continues to issue plans as normal. More statuses of the file replicas can be introduced in order to ensure safe transactions. However, failures of the file catalog service are beyond the scope of this work.

8 Simulations of distributed data production

8.1 Overview of the implementation

In order to validate our new job scheduling approach towards possible use cases, we perform simulations with a wide scope of experimental problems. In the simulations, we measure performance improvement gained by the approach compared to other common scheduling techniques. The simulations are performed with the "Grid Simulation Toolkit for Resource Modeling and Application Scheduling for Parallel and Distributed computing" (GridSim) [209]. It is a Java library for discrete event simulation which provides models of computational servers, job scheduling and execution, networking, data transferring, etc. Previous experience of our colleagues [133] helped to improve efficiency of the GridSim simulations. For our study, we have implemented an additional functionality for plan generation and execution, storage management and statistics collection on top of GridSim. The planner is implemented in Java using JGraphT [210] library which provides graph objects and algorithms. The further described simulations were executed at the Golias computational facility of the Institute of Physics of the Czech Academy of Sciences.

8.1.1 Input data for simulations

The parameters of computational jobs used in our simulations were taken from log records of real data production which was performed for the STAR experiment (US) at KISTI (South Korea) computing facility [204]. During that data production campaign 60,000 of files were processed during three months in 2014 [21]. The average parameters of the set of jobs are provided in Table 3.

Also, the dependence between job parameters is visualized in Figure 9 presented in Chapter 7.1. We have created larger datasets using a random selection from the original one in order to use them as input for our simulations. Such an approach allows us to test the stability of the results against input variation and duration of simulated data production. Also, the larger datasets are required for simulations of large-scale infrastructures. The size and parameters of used datasets are explicitly specified for each set of simu-

Table 3: Parameters of 60,000 data production jobs from the STAR experiment used in simulations.

Parameter	Units	Average	Min	Max	Total
Duration	hours	46	12	64	314 years
Input size	MB	4,320	895	4,780	259 TB
Output size	MB	3,022	105	3,614	181 TB
α	s/MB	38	9	49.2	-
β	-	0.7	0.3	0.79	-

lations in the rest of this chapter.

In order to setup realistic parameters of simulated computational sites and networks, online monitoring tools of CERN experiments [184, 211, 212] were considered. According to these sources, the number of CPUs available at sites varies from several units up to several thousands, while utilized network bandwidth typically is 50 Mbps to 2 Gbps but can reach 10s of Gbps for certain parts of the infrastructure. Since one of the main goals of our approach is to improve utilization of sites with poor network connectivity, in our simulations we have used lower values of bandwidth from the observed spectra. In all the simulations the CPUs are assumed to be of the same processing speed. The initial distribution of data varies in our experiments, it is additionally explained in the description for each particular set of simulations.

Parameters α_i, β and $\widehat{f_{in}}$ of the planner were set to the average values from the Table 3. For the selection of the planning time interval ΔT it is important to notice that it should be short enough to provide a better adaptability to changing states of the resources, but at the same time long enough so that average values and estimations remain applicable. After a set of testing simulations ΔT was set to 12 hours, and the upper limit for the disk usage δ was set to 95 %.

8.1.2 Network models

Let us briefly describe network models provided by GridSim and used in our simulations, more details can be found in [213]. When a simulated entity (i.e. computational site) executes a "transfer file" command, the file is stored to the output queue and then it is processed by other network entities. Each network entity (e.g. router, link) has its own queue. We have used two file transfer models provided by the GridSim framework:

- *Sequential:* Files are transferred one by one in the order as they appear in the queue, only one transfer at a time is performed. This corresponds to modeling a network link as a unary space-shared resource.

- *Parallel:* All the files in the queue are being transferred simultaneously, sharing the bandwidth. In particular, newly started transfers delay those in progress. In this case a network link is modeled as a time-shared resource.

The general behavior of real networks, where many streams of data are transferred concurrently by independent applications, is more realistically described with the parallel model. However, dedicated data transfer tools (e.g. FDT [206], GridFTP [214], BBCP [199]) for transfer of large sets of files can achieve behavior similar to sequential model of GridSim.

Let us illustrate how the difference between the two models can affect distributed data processing. Consider a situation when a set of files is being transferred from one site to another simultaneously by different jobs. In both models, the complete transfer time of the set is the same, but in the sequential one, files will start to arrive earlier. As the result, the processing starts and finishes earlier, releasing resources for the next portions of data, which allows to reduce an overall makespan.

A comparative study of parallel and sequential transfer models in the real network can be found in [215]. The author has shown that transferring files sequentially (but using multiple threads) is advantageous for HENP computations compared to parallel transfer of multiple files. However, the parallel transfer is a more common model for current distributed data processing in HENP. It corresponds to many jobs running independently and performing uncoordinated concurrent access to the remote data over a shared network.

8.1.3 Simulated scheduling approaches

In order to test our planning approach against others, we have simulated distributed data production under the *scheduling approaches* listed below.

- *PLANNER:* This approach uses the planning proposed in our research (described in Chapter 7). The sequential transfer mode is used as the preferable one.

- *PUSHpar:* Whenever there is a free CPU at a remote site, the next input file is sent there from the central storage. When a site receives an input file, it starts processing, and after it is finished it sends the output file back to the central storage. When the central storage receives an output file it sends the next input file to the freed CPU. The process continues until all the data are processed. The shortest network path is used for file transfers. The parallel transfer mode is used here. This scheduling approach corresponds to the distributed data production setup in many HENP experiments, including the data production at KISTI for the STAR experiment [21] in particular.

- *PUSHseq:* The job scheduling is performed exactly as in the previous approach, but the sequential transfer mode is used. The main purpose of these simulations is to study the effects of the sequential file transferring on the data production and, also, to estimate which part of the performance improvement in the PLANNER approach is achieved by the sequential transferring itself.

In addition to that, in experiments with a large number of sites and input sources we have simulated another scheduling policy which we denote as *PULL*. This scheduling policy is currently implemented many HENP experiments. While details of the implementations may differ, the general pattern is very similar: a *pilot job* [4] is submitted to each available CPU in the system which is responsible for requesting input data (pulling), starting computational jobs and transferring output data. When the CPU is ready to process data, the pilot job requests the distributed data management (DDM) system for a new input file. The DDM checks the data availability and redirects the request to one of the sites storing the data. The selection of the site may be arbitrary or depend on either current load or communication latency to the

requester. After the source site is selected the pilot job transfers the data to the local storage and starts to process the job. When the file is processed the pilot job transfers the output to a predefined destination and requests for the next input. As one can see, under such model the CPU allocation and data access are concurrent and uncoordinated. Under the best case scenario all the jobs are transferring data from the fastest available source. We have simulated a pull scheduling approach using the following algorithm executed at each processing site:

- Initialization
    1. Ping all available sources, form a queue ordered by connection speed and set the fastest source as selected for this processing site
- Simulation start
    1. Whenever a CPU becomes free request next input file from the active source
    2. When an input file is received submit a job
    3. When a job is finished transfer the output to its destination
    4. When the current source is depleted, switch to the next one in the queue
    5. Repeat until all the data are processed

Output files can be sent to a single storage, or to multiple ones, using the same reasoning as for input files. The sequential transfer mode is used in the simulations with PULL approach in order to make a fair comparison against PLANNER.

In the case of a single input source and an output destination, both PUSH and PULL approaches result in a similar scheduling as there is no choice of options for the PULL approach. Therefore, in such simulations we compare the PLANNER against PUSHseq/PUSHpar. In other cases, when the input data are initially stored at multiple sites, we test the PLANNER against PULL approach, because it corresponds to the current setup of the largest HENP experiments.

The main metrics used for the performance comparison is the *makespan*, which is calculated as time passed from the start of the first input file transfer (or job submission) until the completion time of the last output file transfer.

To compare two scheduling approaches, a *makespan improvement* for processing the same dataset on the same resources can be calculated as follows:

$$makespan\ improvement = \frac{C^1_{max} - C^2_{max}}{C^1_{max}} \tag{29}$$

8.2 Base model

8.2.1 Single remote site

We start verification of our approach with a relatively simple case where the infrastructure consists of a central storage and a single remote site. Such a setup corresponds to data production of the STAR experiment performed at KISTI computing facility [21]. In the real setup, the only remote site had 1,000 CPUs and the point-to-point network bandwidth was approximately 2 Gbps. Despite our planner is primarily designed to address more complex infrastructures (where many options for optimization exist), consideration of such case is also necessary. First of all, it allows to ensure that the performance of the planner is at least as good as other simple approaches even for a trivial use case. The simulations of the smaller problem allow us to understand the limitations of distributed data production. In particular, we consider the following questions: What is the minimum required bandwidth required to saturate a given number of CPUs at the remote site? How many CPUs at the remote site can be exploited efficiently with a given network bandwidth? How does a selection of a scheduling approach influence those values? The simulations provide answers which can be used when planning future data production campaigns.

Figure 14 present the results of simulations where the number of CPUs at the remote site is 1,000, the size of its storage is 15 TB and the network bandwidth is changing from 50 Mbps to 2 Gbps. The makespan improvement of the planner against other approaches is shown as a function of bandwidth. Each point in the plot is an average of simulations with five different datasets (60,000 jobs in each). The values of the deviations are smaller than 0.07 % which confirms stability of the results, therefore the error bars are not visible in the plot. The plot shows that the PLANNER can provide up to 32 % of makespan improvement compared to PUSHpar approach. At the same time, the difference in performance between the PLANNER and PUSHseq is

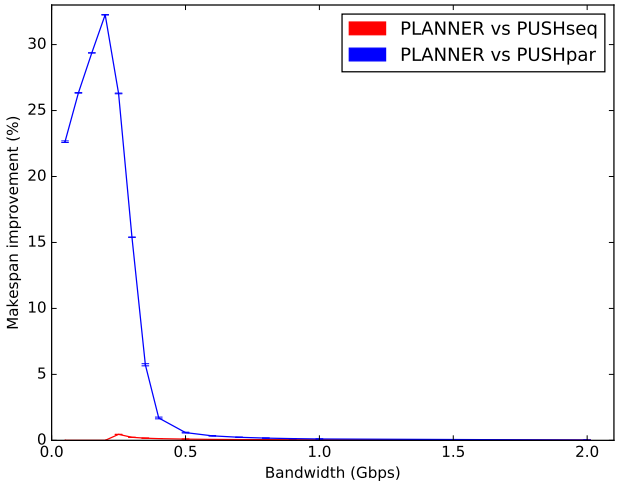


Figure 14: Makespan improvement of the planner as a function of network bandwidth to the remote site.

negligible. This shows that the main part of the improvement is gained due to the sequential data transfer. In parallel transfer mode independent jobs compete for the network bandwidth and as a result the latency increases leading to longer makespan. The difference increases as the available network bandwidth becomes smaller. Additional observations have shown, that the PUSHpar approach fails to utilize 100 % of available CPUs when the bandwidth is below 700 Mbps: a significant fraction of CPU's are waiting for input data. Under PUSHseq approach the earlier transfer requests are not delayed by those arrived later, therefore the fraction of waiting jobs remain smaller. For the PUSHseq approach the bandwidth 300 Mbps is sufficient to keep the number of utilized CPUs close to 100 %. The PLANNER, in its turn, transfers the data to the remote site in advance (whenever there is free network capacity) so that the CPUs can start processing of the next portion of data as soon as they finish the previous one. As the result for the PLANNER the bandwidth of 250 Mbps is sufficient to utilize all the CPUs. It also provides a higher CPU utilization compared to PUSHpar when the bandwidth is below the critical value.

Another set of simulations (see Figure 15) was performed with a fixed network bandwidth (1 Gbps) and a changing number of CPUs at the remote

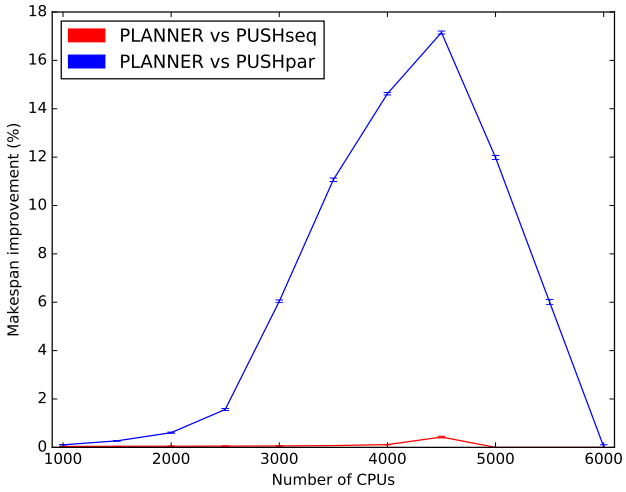


Figure 15: Makespan improvement of the planner as a function of the number of CPUs at the remote site.

site from 1,000 to 6,000 with a step of 500. The size of the local disk was adjusted to the number of CPUs (15 TB for every 1,000 CPUs). Addition of CPUs allows to decrease the makespan proportionally to their number unless the network performance becomes a bottleneck. After the network is saturated the addition of more CPUs does not influence the makespan. However, the maximal number of CPUs which are used to achieve the shortest makespan depends on the scheduling approach and the transfer model. The shortest makespan for all three approaches was approximately 28 days. The PUSHpar simulation reveals the worst performance. Such approach has its shortest makespan reached with 6,000 CPUs and at any given time a significant fraction of CPUs is waiting for input data. As in the previous case, the performance of the PLANNER and PUSHseq approaches is close. Serving data transfer requests in FIFO order provides a significant improvement compared to bandwidth sharing. The shortest makespan was achieved using 5,000 CPUs by both approaches.

We can conclude, that for distributed data-intensive applications over slow networks organized data transfer is advantageous compared to concurrent sharing of bandwidth by many jobs.

8.2.2 Fully connected network

The next set of simulations is dedicated to testing our planner with a more complex and general use case. The computational infrastructure of large HENP experiments consists of many sites. Their connectivity to the central storage (Tier-0) is not always perfect, especially if those are opportunistic resources outside of the primary infrastructure. Let us consider three representative cases: a "fast" site (network bandwidth is not saturated even if all the CPUs are running data production jobs), a "medium" (the available bandwidth is close to the average I/O of running jobs, but not to the peak) and a "slow" (the bandwidth is too small to saturate available CPUs with data, I/O waiting time is increased). Since the sites belong to a complex network, there often exists a path between them which do not share network links with the shortest path to the central storage. Moreover, some of the sites are connected via direct dedicated links. The data can be forwarded between the sites in order to address network bottlenecks. To discover and use such alternative routes the planner considers data movements in the entire network. In this set of simulations we study how the use of such alternative transfer paths can improve computational performance. The infrastructure for these simulations is depicted in Figure 16. It consists of three computational sites named FAST, MEDIUM and SLOW which are connected to the central storage with 1 Gbps, 300 Mbps and 100 Mbps network links, respectively. In addition to that, all the sites are interconnected with links of equal capacity (the dotted lines in Figure 16). The capacity of those links

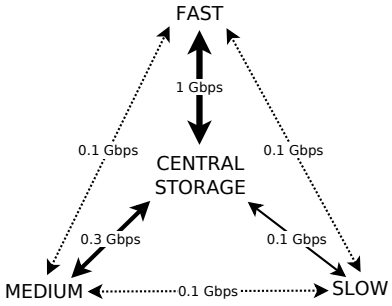


Figure 16: Simulated infrastructure with a fully connected network topology.

is changing from 10 to 500 Mbps in different simulations in order to study

how the use of alternative transfer paths can improve the data production efficiency. Each site has 1,000 of CPUs and a local disc of 15 TB size.

The results of the simulations are given in Figure 17. The plot shows how

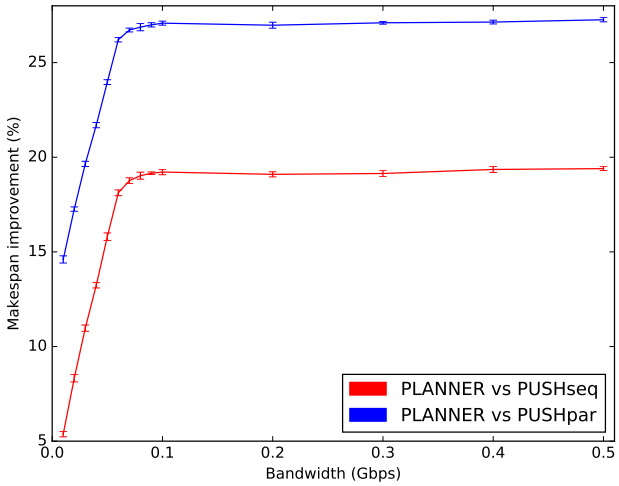


Figure 17: Dependence of makespan improvement on bandwidth between remote sites.

the makespan improvement of the PLANNER compared to PUSHpar and PUSHseq depends on the bandwidth of the links between the remote sites. Each point in the plot is an average of five simulations with different datasets (60,000 jobs in each), and the error bars represent a standard deviation. The planner has reached up to 27 % improvement against the current scheduling approach (PUSHpar) and up to approximately 19 % improvement against the PUSHseq. Additional monitoring has shown that the planner, as expected, redirects a part of the input flow from the "FAST" site to the "MEDIUM" and "SLOW"; and from "MEDIUM" to "SLOW". This allows to balance the network load in order to avoid congestion and achieve higher CPU utilization.

Figure 18 shows the CPU utilization (percentage of busy CPUs over the total number of provided CPUs) as a function of time for one of the simulations with 100 Mbps interconnecting links. As it can be observed, both PUSHpar and PUSHseq models did not manage to utilize 100 % of CPUs, and the number of busy CPUs is fluctuating over time. At the same time, the

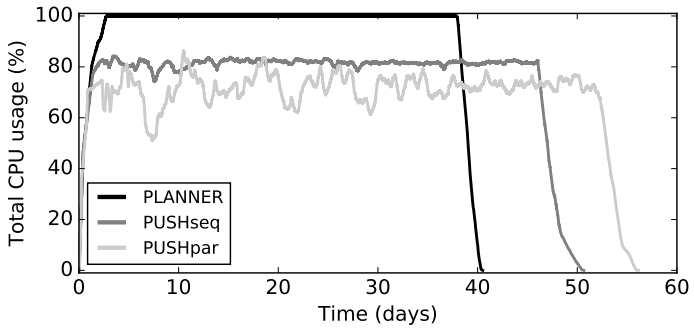


Figure 18: Total CPU usage in the simulation with 100 Mbps links between remote sites.

PLANNER reaches 100 % CPU utilization shortly after the start and maintains it until the end of data production. This is achieved due to distributing the network load over the links which would be idle otherwise.

In these set of simulations an average makespan improvement of PUSHseq over PUSHpar is 10 %. As in the previous simulations with a single link, sequential transfer mode appeared to be advantageous. More importantly, these simulations have demonstrated that our planner can efficiently schedule data transfers in a network and decrease processing makespan. Compared to the approach which also uses the sequential transfer model but relies on the shortest path only (PUSHseq) our approach provides an improvement which is up to 19 % for the simulated infrastructure.

8.2.3 Random scale-free networks

Scale-free topology is often used to present common properties of communication networks [216]. In this set of simulations, we test our planner over a wide set of arbitrary infrastructures. Each infrastructure consists of 30 sites interconnected with a random scale-free network. The network bandwidth, amount of CPUs and storage were down-scaled compared to real facilities, in order to keep the complexity of the simulations within a reasonable limit. This also illustrates the fact that only a fraction of overall resources are typically available for data production (the rest are used for other activities e.g. user analysis and simulations). To generate a random infrastructure we use

the following steps:

1. Generate a random scale-free network of 30 sites.
2. Set the site with the highest network degree as a Tier-0 (the only input source and output destination). No CPUs at Tier-0 are considered.
3. For the rest of the sites, set the number of CPUs between 100 and 200 at random. The storage size is 30 GB per each CPU.
4. For each link set a random bandwidth between 0.05 and 0.15 Gbps.

20 distinct infrastructures were generated and used for the simulations. Several examples are provided in Figure 19, where the site's size is proportional

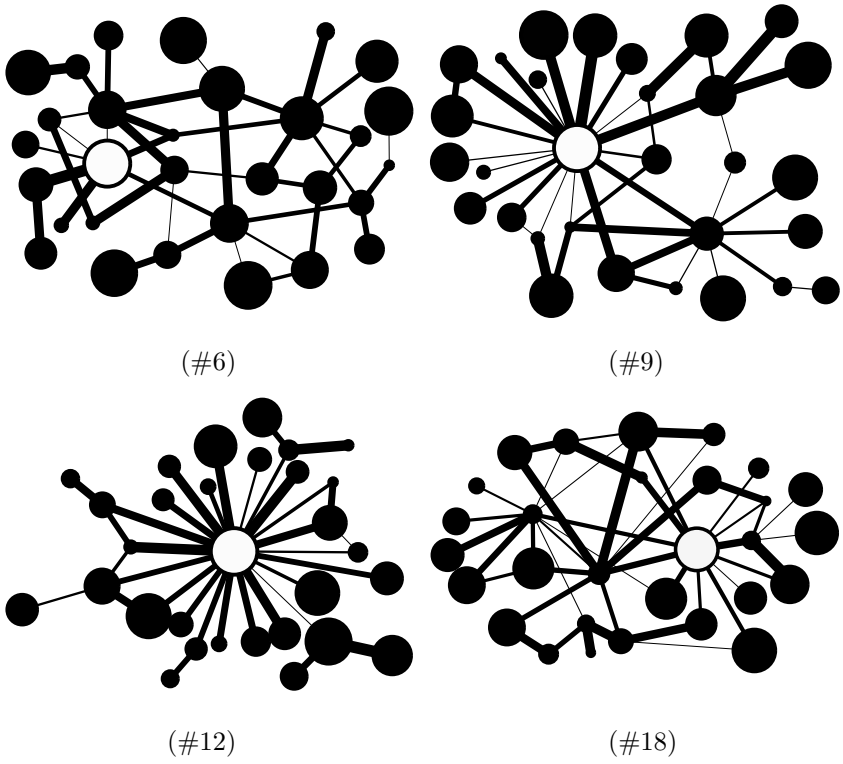


Figure 19: Examples of randomly generated infrastructures.

to the number of CPUs, link thickness is proportional to the bandwidth and the Tier-0 site is colored in white. We simulated data production at

each infrastructure using 3 scheduling approaches (PLANNER, PUSHpar and PUSHseq) and 5 datasets (60,000 jobs each).

Figure 20 shows the makespan improvements our PLANNER provides

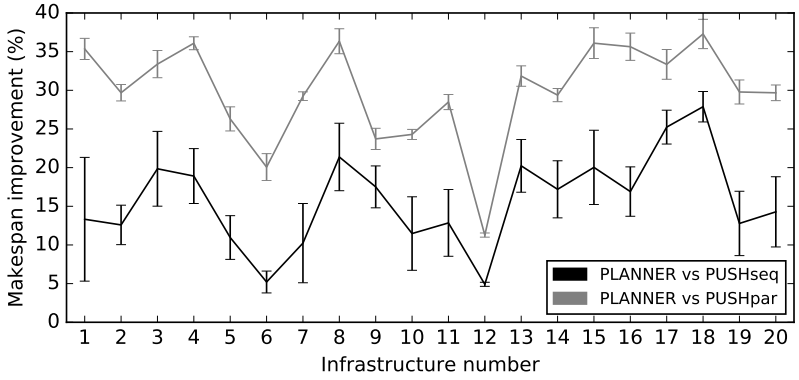


Figure 20: Makespan improvement of the planer in randomly generated infrastructures.

compared to two other approaches for each of 20 generated infrastructures. For each infrastructure the results are averaged over 5 simulations with distinct datasets, the error bars show the standard deviation. In all of the simulations, the planner has consistently demonstrated a positive improvement compared to both PUSHseq and PUSHpar. The magnitude depends on the properties of a particular infrastructure: presence of bottlenecks and opportunities to automatically mitigate their influence. The makespan improvement by the PLANNER is within 5 – 27 % and 11 – 37 % compared to PUSHseq and PUSHpar, respectively. As before, the organized (sequential) transfer execution appeared to be more efficient than uncoordinated (parallel) one.

8.2.4 Real infrastructure

In order to test the scalability of the planner, we have performed simulations using data on Tier-1 sites of one of the currently largest experiments in HENP. We have used statistics of 10 computational sites obtained from online monitoring tools. However, the designed parameters of such a network

are well balanced and a transfer latency does not influence the computational performance significantly. Under these conditions, all the simulated approaches provide similar performance. For a better illustration of the planner abilities, we have added a dummy site called "PLUTO" to the initial grid. The site has a significant amount of CPUs but a limited connection (1 Gbps) to the Tier-0 while it has a reasonable connection to two other Tier-1 sites. Such situation can often emerge in real life, when there are computational resources provided at sites outside of the CERN network infrastructure. The final grid setup used in the simulations is provided in Figure 21. The ag-

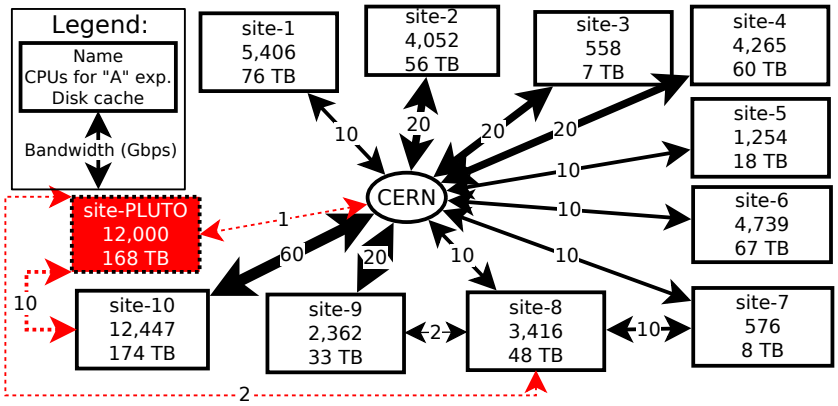


Figure 21: Simulated grid of Tier-1 sites of one of the largest HENP experiments.

gregated number of simulated CPUs is 39,075. The amount of disk space at sites was set proportional to the number of CPUs since we assume that only a part of the overall storage can be dedicated to data production. For these simulations we used a set of 500,000 jobs which was generated using random selection from the original set.

The plot in Figure 22 shows how the total CPU utilization in the simulated grid changes with time for PUSHseq and PLANNER scheduling approaches. Unfortunately, the results for PUSHpar approach are not presented here as the underlying GridSim model for the parallel network mode appeared to be too computationally demanding. However, this approach has shown the worst performance in all previous simulations. As one can observe in Figure 22, the PLANNER has reached 100 % of the overall CPU performance

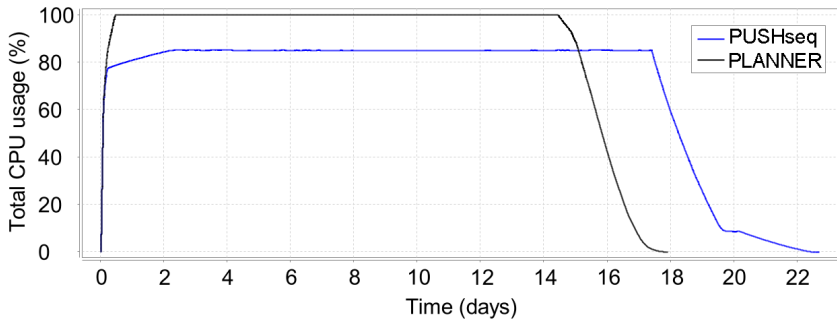


Figure 22: Results of simulations of the realistic Tier-1 grid.

since it was able to utilize all the CPUs at the "PLUTO" site using alternative (indirect) transfer paths. Such approach allows to use more computational resources outside of the primary network infrastructure. The makespan improvement in the considered case is 21 %, an increase in throughput which would not have been easily possible without reasoning and a planner.

During the simulations with the realistic size of grid, an average runtime of the planner was measured. The result shows that planning for 12 hours of data production is done within 7 milliseconds. This allows to apply this approach for online planning of data production in the real environment.

8.3 Influence of background network traffic

Network infrastructure is often shared between several experiments, as a result, transfer latency may increase when several network activities are ongoing simultaneously. We have studied the influence of the background traffic on the efficiency of data production comparing different scheduling approaches with the help of simulations. The simulated infrastructure consisted of a single remote computational site with 1,000 CPUs and 14 TB local disk connected via 1 Gbps network link to the central storage (Tier-0). Both sites were sending unrelated files of 12 GB size every 1,000 seconds to each other, the number of those files to be sent at once was changing in different simulations. Since simulations with background traffic appeared computationally challenging in GridSim, we used a smaller set of jobs. The same set of 7,000 jobs from the original set of the log records was used in each simulation. The performance of the simulated scheduling approaches is com-

pared in Figure 23. As one can see in the plot, our PLANNER shows better

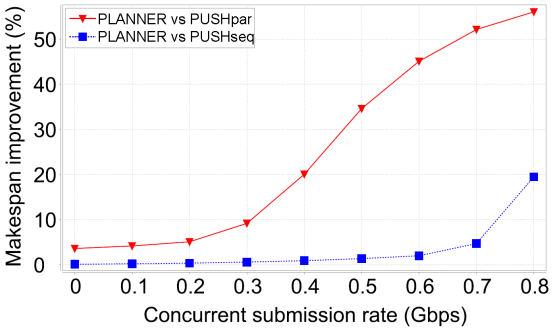


Figure 23: Results of simulations with background traffic.

performance in all the simulations and the gain in performance increases as the level of background traffic grows. The PLANNER's makespan increases just by 0.17 % when changing from 0 to 0.8 Gbps of the concurrent transfer submission rate while the makespan for the PUSHseq and PUSHpar is much higher corresponding to 24 % and 120 %, respectively. This is achieved due to file transfers in advance before computation, so that the jobs are not delayed by the network latency.

It is important to consider the maximal possible network latency when setting up the planning time interval ΔT. If ΔT is too small, the transfers started at one planner cycle may be completed at the next one and that can compromise the preciseness of the plan execution. The simulations have shown that ΔT set to $6-12$ hours is large enough to avoid such issues.

8.4 Multiple input sources and arbitrary networks

8.4.1 Simulated infrastructure

Here we simulate data production in arbitrary large-scale heterogeneous infrastructures where data are distributed between multiple sites. The infrastructures are generated using random scale-free networks [216]. We have modified the generation procedure described in Section 8.2.3 in order to impose additional features common for data grids. For instance, in HENP computing (see Section 5), the data are typically placed at regional centers (Tier-0/1) which have better connectivity to each other than the rest

of sites. Also, larger computing facilities typically have a better network connection and many smaller sites are connected through them. In order to model such properties, we set the link bandwidth, number of CPUs and storage size to correlate with the network degree of a site. The number of CPUs is defined as a product of two factors: network degree and random. The latter one is needed to increase heterogeneity of the system. The storage size is set proportional to $NCPU_i$. The link bandwidth is a product of a *reference bandwidth* and the smallest values of network degree and NCP_i of connected sites. We vary the reference bandwidth in order to generate infrastructures with different network performance. However, we set the limits for the reference bandwidth and the random factor for CPUs in such a way, that the resulting infrastructures would comply with observations from the online monitoring data of HENP experiments (such as MonAlisa [184]). The resulting procedure is described in the following way:

1. Generate a scale-free graph with N vertices.
2. M vertices with the highest degree are configured as input sources.
3. The rest of the vertexes are configured as processing sites, where $NCPU_i$ is proportional to the vertex degree and a random value. The size of the local disk is set to 15 GB per CPU.
4. Each edge of the graph is set as a network link. Its bandwidth is defined as a reference bandwidth multiplied by the smallest $NCPU_i$ and network degree of the sites connected by the link.

Thirteen generated computational networks used in the simulations consisted of 30-50 sites and were varying in a total number of CPU's (4,000 – 23,000), network structure and bandwidth of the links. Figure 24 shows an example of such network with 50 sites, 8 input sources, 77 links and 6,663 CPUs. The red vertexes at the picture are input sources, the blue vertexes are processing sites, where the label is the number of CPUs. The thickness of the links illustrates the bandwidth.

Four simulations were done for each generated network: PULL and PLANNER using multiple input sources; and PULL(single) and PLANNER(single) using a single input source. In case of multiple input sources, the initial data distribution was established using our approach described in Chapter 7.5. In

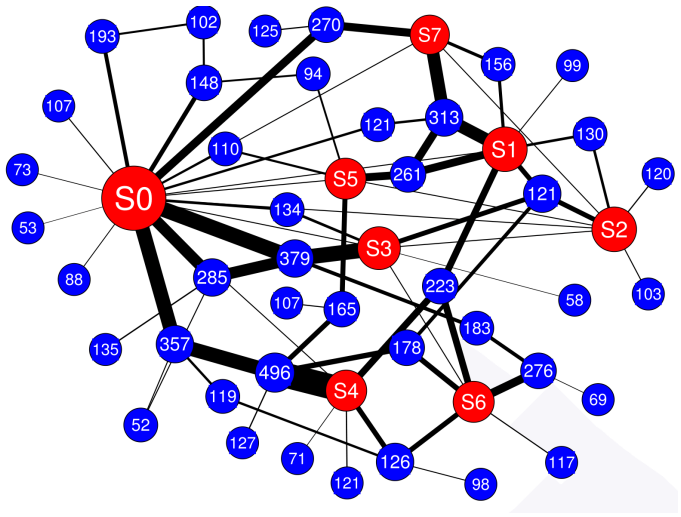


Figure 24: Example of a randomly generated infrastructure with multiple input sources.

case of a single source, the site with the best connectivity (largest degree) was used. S0 site in Figure 24 is an example. In both cases, the output data was transferred to a single output destination which was, again, the site with the best connectivity (S0). In each simulation, the same dataset consisting of 60,000 jobs (259 TB of input data) was submitted for data production, and the resulting makespan and CPU utilization over time were compared.

8.4.2 Results

In all the simulations the PLANNER has reached the highest CPU utilization (both peak and average) and a significantly shorter makespan. The results of the simulations are summarized in Table 4. For each setup (denoted in the first column) the table provides parameters of the infrastructure, initial data distribution planned by our approach (percentage of the entire dataset placed at each input source), makespan of the PULL(single) approach and the makespan improvement of other approaches over it. Figure 25 (parts 1 and 2) contains plots of total CPU usage over time for each scheduling approach in each setup. Under the PULL model a small fraction of jobs which is processed the last increases the overall makespan dramatically. This can be seen as a long "tail" for both PULL and PULL(single) threads in the plots. It is a well known drawback of remote data access in scale-free networks. Such behavior

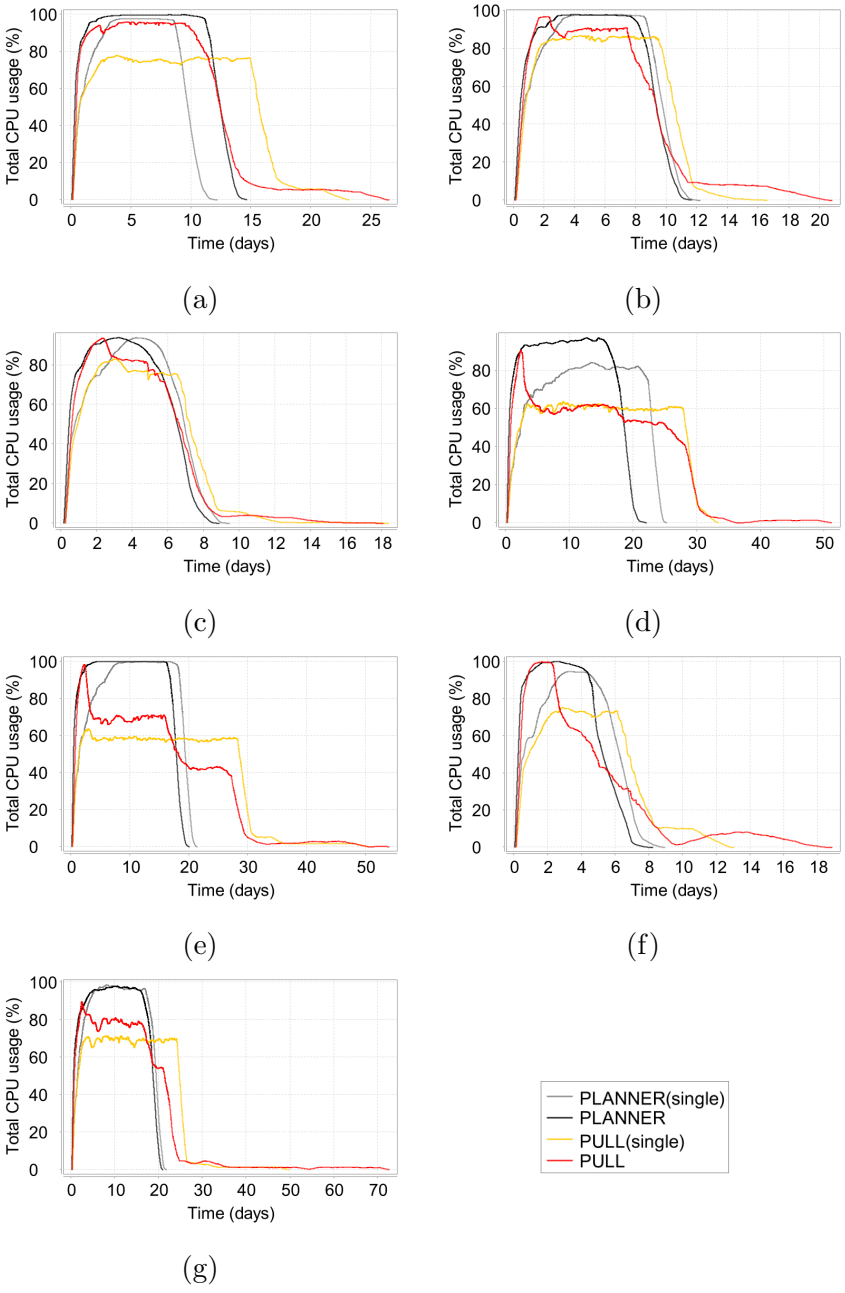


Figure 25: (Part 1) Results of simulations with randomly generated infrastructures and multiple input sources.

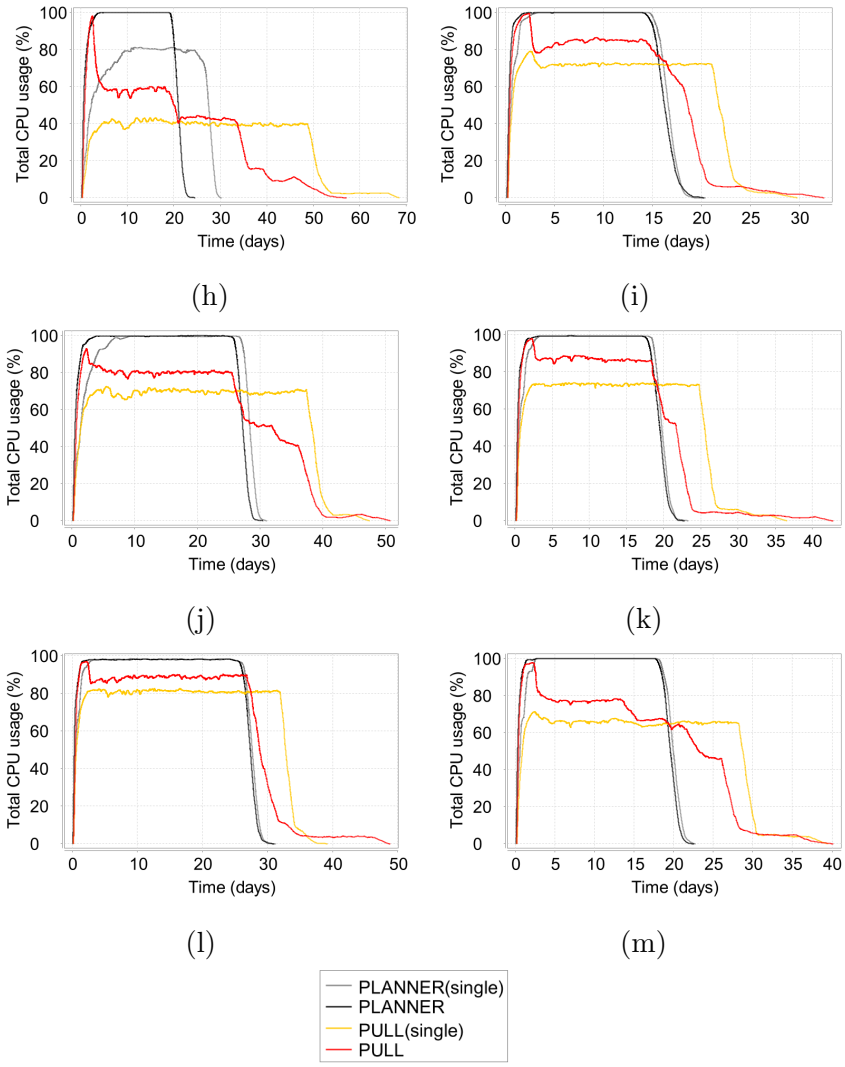


Figure 25: (Part 2) Results of simulations with randomly generated infrastructures and multiple input sources.

Table 4: Results of simulations with randomly generated infrastructures and multiple input sources.

#	Sites	Sources	Links	CPUs	Reference bandwidth kbps/CPU	Planned initial data distribution %	Makespan PULL (single) seconds	Makespan improvement PULL %	PLANNER (single) %	PLANNER %
a	30	6	41	9,609	300	55, 30, 8, 3, 2, 2	1,993,700	-14	34	37
b	40	5	54	13,425	300	65, 12, 20, 3, 0	1,422,138	-25	26	29
c	50	5	69	20,982	300	57, 14, 5, 12, 12	1,571,094	2	49	52
d	50	5	81	6,771	200	44, 34, 16, 6, 0	3,927,620	-12	44	51
e	50	8	77	6,663	250	45, 20, 4, 1, 15, 6, 9, 0	4,346,099	-6	57	60
f	30	5	42	22,463	500	44, 41, 15, 0, 0	1,116,891	-45	31	37
g	50	5	72	6,717	200	56, 10, 10, 15, 9	4,275,359	-46	56	58
h	50	5	72	5,704	300	29, 16, 29, 14, 12	5,893,276	15	56	64
i	50	5	91	7,210	300	47, 38, 8, 2, 5	2,554,741	-9	31	32
j	50	5	69	4,327	300	48, 10, 19, 21, 2	4,087,063	-7	35	36
k	50	5	73	6,084	400	66, 15, 7, 3, 9	3,140,344	-17	36	38
l	50	5	73	4,342	400	69, 15, 8, 1, 7	3,381,916	-24	20	21
m	50	5	75	5,970	500	33, 39, 16, 2, 10	3,397,126	-2	42	43
						Average makespan improvement:		-15	40	43
						Standard deviation:		17	12	13

can be explained by the lack of coordination between CPU allocation and file transferring: the jobs are allocated to the first CPU which becomes free with no reasoning about the resulting latency. As a consequence, significant portions of data are sent to distant (in a network sense) processing sites, especially at the end of the computation. At the same time, the closer sites run out of input data and remain idle. In contrast, the PLANNER considers how much data can be transferred and processed at each site within each planning time interval and distributes the load accordingly. This allows to decrease the makespan dramatically: by 43 % on average with deviation 13 % in all the simulations (PLANNER compared to PULL(single), please, note, that PULL(single) has better performance than PULL as discussed later). Such a significant makespan improvement has a great value for applications where the dataset has to be processed completely before its future usage can start (e.g. user analysis in HENP processes the outcome of data production).

The initial data distribution helps to reach the peak processing throughput faster (compare PLANNER and PLANNER(single) in Figure 25 parts 1 and 2) and thereby provides an additional decrease of the makespan by 6 % on average compared to a single input source. It becomes even more advantageous (up to 18 %) if the network has regions with poor connectivity

to the primary input source as in cases d, f and h. In such case, transferring a portion of input data to that region before the processing starts allows to utilize resources more efficiently. However, our initial data distribution does not bring any advantage for the PULL approach, because this approach does not address well the case of multiple sources without data replication. When multiple sources contain unique portions of data the transfer latency increases for the PULL algorithm at late stages (when only sources with average connectivity left) compared to the case with a single source (with the best connectivity). Let us note that most of the modern DDM systems can provide data replication across sites. It allows to select a closer input source for each particular job, which is beneficial for the PULL model. We consider data replication in the next section.

8.5 Data replication

8.5.1 Simulated infrastructure

In this section our simulations are focused on another extremely important features — data replication and heterogeneous infrastructure composed of Tier-0,1,2 sites. We took the structure of the Tier-1 network of a real LHC experiment (see Section 8.2.4) as a core. Since the detailed network data for Tier-2 sites was not available, we added those sites randomly, using scale-free algorithm applied in Section 8.4.1. The resulting procedure was the following (see Figure 26):

- Ten Tier-1 computational sites (B0 – B9, colored magenta) and the primary network infrastructure of one of the largest HENP experiments are used as a core. The number of CPUs, storage size and network bandwidth are downscaled by the factor of two. This allows to run the simulations within a reasonable time. The total number of simulated CPUs at those ten sites is 19,536. We assume that each Tier-1 site initially stores a partial replica of input data (replicated from Tier-0). The replicas are of an equal size and non-overlapping.

- Another Tier-1 site (B10, colored blue) with a significant portion of CPUs (6,000), slow connection to Tier-0 and no locally stored input data is added to the system in order to challenge compared schedul-

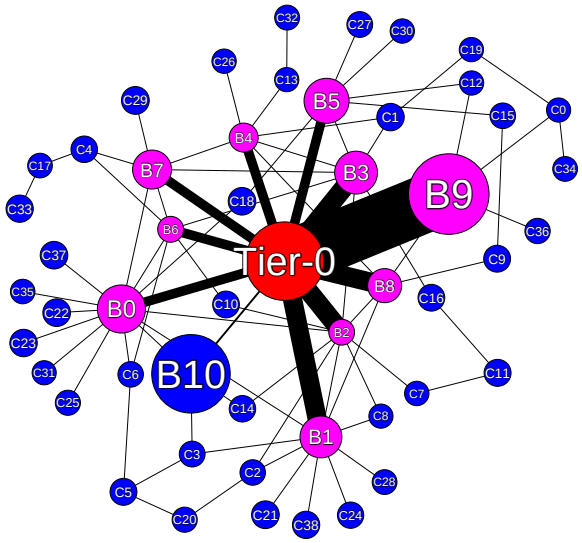


Figure 26: Simulated infrastructure with data replication.

ing approaches. The site represents a large computing facility which is outside of the primary network infrastructure. Such case can illustrate the usage of opportunistic resources or resources of an external volunteering organization possibly from a distant part of the globe.

- A single Tier-0 site (colored red) is considered as a storage only, i.e. it is another source of input files and the only destination for output files. We assume that the Tier-0 site persistently stores the entire input dataset, and all the output files have to be copied there. In many of the experiments, the Tier-0 has a significant computational power and is often used for data production. However, processing data at the Tier-0 site does not require remote data access and, therefore, is removed out of the scope of our simulations. This simplification decreases the number of simulated CPUs and, therefore, reduces the complexity of the simulations. Nevertheless, our approach can include data processing at Tier-0 site with just a trivial change in configuration.

- The number of CPUs and storage size at Tier-2 sites were assigned randomly with respect to values observed in real infrastructure ($\sim$100 CPUs, $\sim$ 3 TB). The sites were connected to the rest of the infrastructure using an algorithm for a scale-free network generation [216].

The bandwidth of the corresponding links was set within 0.1 – 1 Gbps. The simulated infrastructure contains 39 Tier-2 sites (C0 – C38, colored blue) with 11,019 CPUs in total.

The resulting computational network used in our simulations consists of 51 sites with 36,555 CPUs in total and 82 network links. At Figure 26 the size of the nodes is proportional to the number of CPUs at the site, thickness of the edges is proportional to the bandwidth of network links. Five different datasets were created using random selection from the original set of log records of data production of the STAR experiment. Each dataset consists of 600,000 input files with total size of 2.7 PB on average. The corresponding average total size of output files is 1.8 PB. The total time for sequential processing of such dataset is 3,129 CPU years on average. In these simulations the PLANNER was compared against the PULL.

8.5.2 Results

The simulations with the described infrastructure have shown that comparing to PULL the PLANNER allows to process each given dataset with a makespan shorter by 7 % on average with deviation 0.8 %. A typical dependence of the total CPU usage over time during the data production for both PULL and PLANNER is compared in Figure 27. As one can see on the

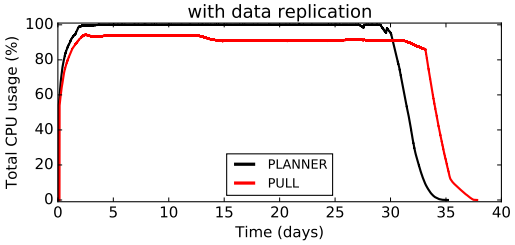


Figure 27: Results of simulations with data replication.

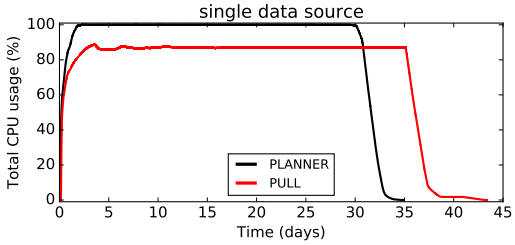


Figure 28: Results of simulations without data replication.

plot, the PULL model fails to utilize all the available CPUs efficiently in the simulated case: a significant fraction of CPUs is waiting for input/output transfers and, for this reason, the actual computations are delayed. In such situation, when the network performance becomes a bottleneck, uncoordinated concurrent data transfer by multiple processes becomes inefficient as

it leads to congestion and increased latency. In contrast, the PLANNER transfers balanced portions of input data in advance before their processing, output transfers do not delay CPUs and the network usage is planned in order to avoid congestion. This allows to reach the maximum CPU usage which otherwise would be hindered by the limited network performance.

For better illustration, we provide CPU usage per site for PULL and PLANNER in Figures 29 and 30 respectively. Each plot shows the perfor-

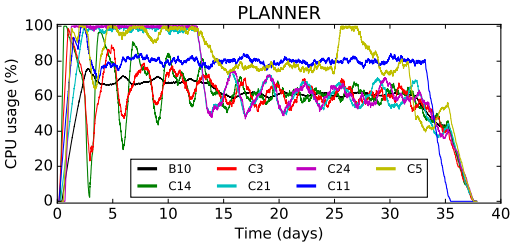


Figure 29: Results of simulation with PULL approach: 7 sites with the lowest CPU usage.

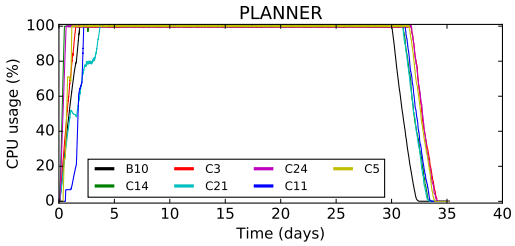


Figure 30: Results of simulation with PLANNER approach: 7 sites with the lowest CPU usage.

mance of seven sites with the lowest CPU utilization under the PULL approach. Sites, such as C3, C11 and C5 (see Figure 26), share network links for data access. As a result, simultaneous I/O access saturates the network capacity and the CPU usage at those sites is degraded. C21 and C24 switch to distant input sources when the closest ones are depleted which increases the latency. Sites, such as B10 and C14 do not have enough connection bandwidth to keep all their CPUs utilized. Despite alternative input sources and transfer routes are available for those sites, the PULL model does not distribute (balance) network load across them. Using the PLANNER, the transfers are distributed in time, between possible sources and over alternative routes in order to match the network capacity. This allows to transfer the data to/from the bottleneck sites efficiently.

In order to study the advantage of data replication, we have performed an additional simulation where all the input data are initially stored at the Tier-0 site only. The results are presented in Figure 28. Comparing it to Figure 27 one can conclude that for the studied use case the PLANNER provides a close computational efficiency with and without data replication. This result can be understood considering that our PLANNER creates tem-

poral copies asynchronously to the data processing that is, moves data closer to the resources as the CPUs are busy for later consumption. The time for moving files being less than the processing time (and the storage capacity allowing the data copies), the difference between replication and no replication is not seen, illustrating the importance (for this use case) of networking bandwidth as a resource. In contrast, the data replication is required to increase the performance of the PULL for two reasons. First, it allows to increase peak performance due to a better distribution of the network load. Second, without data replication, the PULL had a noticeable makespan increase. This delay is due to a small fraction of slow jobs, which are executed at large network distances from the data placement and, therefore, suffer an I/O overhead. Those jobs are observed as a "tail" at the end of the plot in Figure 28. Such effect is reduced with data replication where jobs have a choice of input sources and use the one with the fastest connection. Nevertheless, the PLANNER has shown advantage both with/without data replication. The planner considers how much data can be processed during the planning time interval and distributes data and jobs accordingly: if the data cannot be processed at a remote site faster than at its current location it will not be sent there. Therefore, the overall makespan is not compromised by a small fraction of delayed jobs. In this simulation the PLANNER provides 19 % of makespan improvement compared to the PULL. Similar observations were made in the simulations from the previous Section 8.4.2.

8.6 Computational time

In all the simulations presented in Section 8 we have measured the time required by our planner to produce a solution. In the simulations using base model the average solving time is 7 ms. In more complex simulations with many tens of sites (Sections 8.5.2 and 8.4.1) the average solving time is 30 ms with deviation 14 ms for 500 planner runs analyzed. Since the planning time interval has the order of magnitude of hours, no additional optimization of the planner regarding the solving time is necessary. The short solving time is achieved due to the efficient underlying model of the problem which allows to apply a network flow maximization algorithm. As the scale of the simulated computational network compares to real systems in HENP, such solving time meets the requirements for planning in real infrastructures.

8.7 Summary of simulations

In this chapter, we have presented simulations of distributed data production based on data from real systems. A wide scope of use cases was studied including large-scale heterogeneous infrastructures and shared networks with high background traffic. In these simulations our planning approach was compared against the approaches currently used in the field. The simulations have shown a consistent advantage of the planner which provides higher CPU usage during computations and shorter makespan. The observed values of makespan improvement vary in a wide range and should be considered in the context of the simulated use cases. In small and fast (compared to processing throughput) networks all the simulated approaches have shown close performance. In case of complex infrastructures, network bottlenecks and high background traffic the planner usage becomes clearly beneficial providing 7 – 60 % shorter makespan.

As the planner adjusts data distribution during computations the initial data distribution has a minor influence on its performance (in contrast to the PULL approach) in the simulated use cases. Initial data distribution established using our method can provide an additional makespan improvement and increase efficiency of CPU usage at an early stage of data production. It has shown a stable positive effect which varies from 1 to 18 % with an average of 6 % of makespan improvement, depending on the network properties.

The plan generation takes 30 ms on average for the largest simulated infrastructures which match the complexity of real systems. Since the planning has only to be repeated every 6 – 12 hours, such solving time leaves reserve to consider even larger (prospective) systems.

Overall, we can conclude, that according to the experiments our approach provides optimization for distributed data production in complex networks while no drawbacks were observed for fast and/or simple networks.

9 Cache management for distributed data storage in HENP

Management of distributed data storage with a high degree of replication raises an important question of efficient space usage at sites. Our job scheduling approach described in Chapter 7 assumes temporal file replicas to be removed from a computing site as soon as its processing is finished. However, if there is enough space, it is beneficial to keep data for further reuse. In case of input (raw) data, such retention would speed the data access up in case of future (repeated) data production campaigns. The advantage of keeping more replicas is even more obvious in case of output (reconstructed) data as those files are typically accessed multiple times by many analysis jobs. After the files are transferred to their destinations or processed, temporal copies remaining after the plan execution may be used to improve data availability. For this reason, storage clean-ups can be performed asynchronously from data transfer/processing in order to keep potentially reusable data as long as possible but remove it upon demand for free space.

A storage of a HENP computational site accommodates data related to several types of computing activities: data production, user analysis and simulations. It can be seen as a cache where a part of an entire dataset of a collaboration (experiment) is placed. If a file requested by a job is missing at the local storage it has to be accessed over an outer network which introduces additional latency. Another case is a large data center with a hard drive array backed with a tertiary storage. The hard drive arrays provide much faster data access than a tertiary storage, but they have larger cost, therefore, the available space is limited. For example, in large collaborations, several Tier-1 sites permanently store significant fractions of the entire dataset (100 % in case of Tier-0) on tape and serve data requests from other sites. The data from a tertiary storage is recovered to a hard drive upon request and then sent over the network. If the requested data are already on the disks an overall latency is significantly smaller. To summarize, the local hard drive array at a computing site can be considered as a cache for both outgoing data requests of locally running jobs and incoming data requests of remote jobs. Also, the concept of caching is utilized in data transfer tools such as Falcon [11].

When managing the content of cache it is desirable to keep the data which are the most likely to be accessed in future and delete the rest when space is needed for new data. Plenty of existing cache policys address this task. However, those algorithms are designed to match particular applications (CPU registers, RAM, file systems, web etc.), therefore, a careful study is needed to find the appropriate one for a given use case. In this chapter, we compare performance of known caching algorithms with respect to data access patterns observed in HENP computations.

In this study, all the caching algorithms were implemented following the concept known as "water-marking". Water-marking is an approach where thresholds are set for the cache cleanup starts and stops. It considers the current disk space occupied by the data in cache while the high-mark and the low mark for the cache size can be set up as needed. When the used cache size exceeds the high-mark, the cache clean-up starts, and files are deleted until the used cache size gets below the low mark. The time interval between clean-ups depends on the combination of high/low marks, cache size and data-flow. Therefore with watermarking concept more computational demanding algorithms can be implemented as the cleanup may be independent of data transfer. In heterogeneous grid the size of storage at sites varies significantly: it can be comparable to the entire dataset at Tier-0 site, or can make up just a fraction of a percent at small Tier-2 sites. The size of cache influences its performance and is also considered in this study.

9.1 Data access patterns in HENP

In order to study caching, several data access patterns were extracted from log files of data management systems of HENP computing sites. Three different access patterns were used as input for our simulations:

STAR1: the pattern was extracted from Xrootd [65] log taken from the Tier-0 site of the STAR experiment (RCF@BNL). It consist of access records made during a 3 months period (June – August 2012) for all data available in STAR.

STAR2: the pattern was extracted from the same source (Xrootd of RCF@BNL) but within a different time period: 7 months in August 2012 – February 2013.

GOLIAS computing farm is a part of regional computing center for particle physics at the Institute of Physics (FZU) in Prague and a part of a Tier-2 site for the ATLAS experiment. The facility also performs data processing for another experiment — AUGER, which makes less than 1% of the total requests. The pattern was extracted from DPM [197] log for a 3 months period (November 2012 – February 2013).

The usage of access patterns corresponding to different time periods and experiments allows to verify consistency of the results. The parameters of the three access patterns are given in Table 5. Both STAR access patterns

Table 5: Summary of three user access patterns used in simulations.

		STAR1	STAR2	GOLIAS
Time period	months	3	7	3
Number of requests	$\times 10^6$	33	52	21
Data transferred	PB	50	80	10
Maximal number of requests per file	–	192	203	94,260
Average number of requests per file	–	19	15	5
Number of unique files	$\times 10^6$	1.8	1.7	3.8
Total size of dataset	PB	1.45	2	1
Maximal file size	GB	5.3	5.3	18
Average file size	GB	0.8	1	0.3

have similar parameters. It is noteworthy to mention that the first one was taken right before the Quark Matter 2012 conference and the second one, right after. It is important as the user analysis intensifies before a conference and without verification, it would be doubtful if our findings would be stable over time. The number of files requested only once during the studied period, is less than 10% in both patterns.

The GOLIAS access pattern is taken from another experiment with different data-storage structure, DPM is used here within a Tier-2 data access context (user analysis). This access pattern is much less uniform and differs from the other two: the size of files is not explicitly limited and can reach 18 GB, the number of requests per file varies from 1 up to 94,260, with an average 5. In this dataset, 44 % of the files were requested only once.

When analyzing an access pattern one can subtract a set of unique filenames. It is a set of all files requested at least once during the period of consideration. The histograms at the Figure 31 represent the distribution of

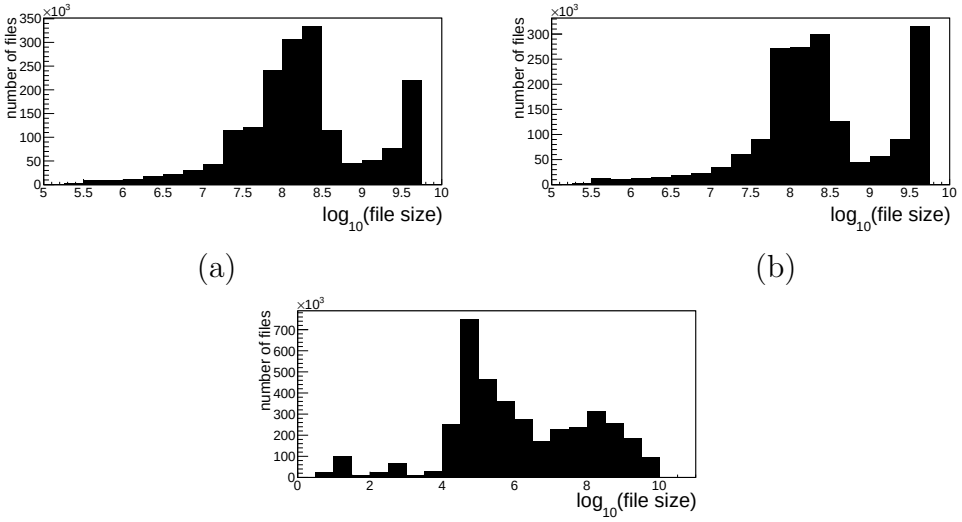


Figure 31: Distribution of files by size for three access patterns: (a) STAR1, (b) STAR2, (c) GOLIAS.

those unique files by size for each dataset. Here one can see that the file size distribution at GOLIAS is more dispersed than in STAR. Also, as it can be observed at the histograms, in STAR maximal file size is limited to 5.3 GB (the files of larger size are split into several files). This fact explains the second peak at the histograms for STAR1 and STAR2 datasets. At GOLIAS there is no limitation for the file size, two peaks at the histogram can be explained with the presence of files with different types of data.

Another important characteristic of an access pattern is a distribution of a time interval between two consequent requests for the same file. These histograms are given in Figure 32. In both STAR access patterns the distribution is close to log-normal with the peak time interval corresponding to 24 hours. This can be explained by the behavior of users. One can imagine a situation when a scientist checks a result of computational job in the morning, edits the code and then resubmits the analysis on the same dataset, and the new output will be available by the next working day. The GOLIAS

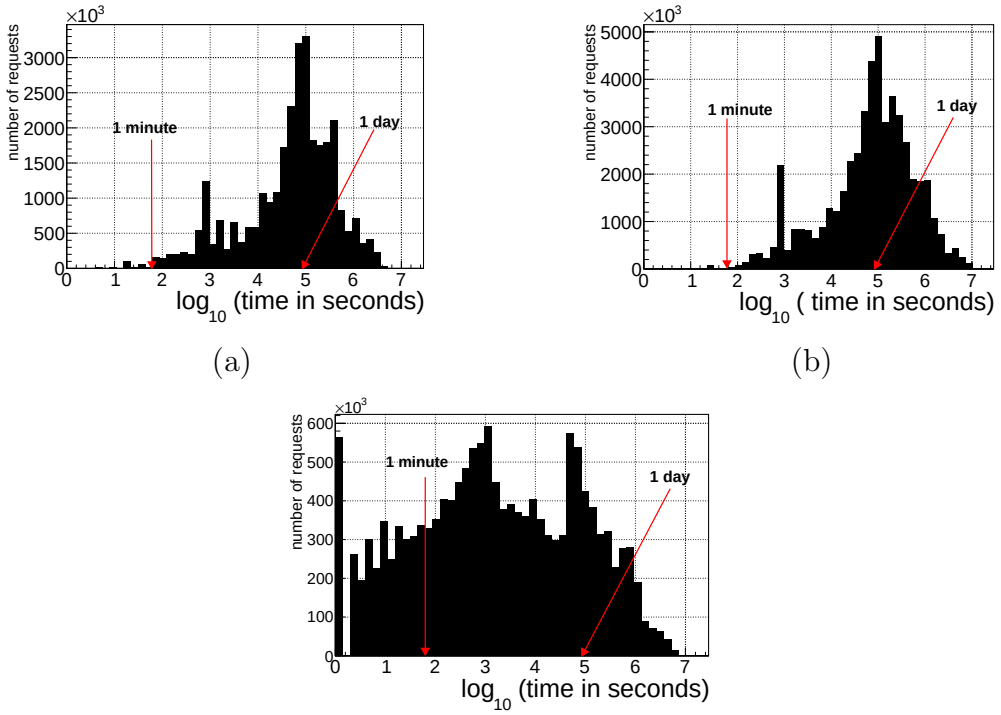


Figure 32: Distribution of time intervals between sequential requests for the same file in three access patterns : (a) STAR1, (b) STAR2, (c) GOLIAS.

access pattern is less regular. This can be explained with the large amount of jobs submitted automatically with different intervals, and different type of jobs having various duration.

Each access pattern can be represented as a contour plot (see Figure 33) where axes are the number of requests for a particular file and its size, color represents the number of files with the same coordinates on the plot. The hot spots on the contour plot allow to reveal common cases of file usage. In all three access patterns the densest area corresponds to small files that have been accessed several times only. In both STAR patterns multiple other hot spots are visible. For example, a set of large files (average 4.8 GB) which makes 6 % of the entire dataset by number and 37 % by size is responsible for 15 % of requests and 47 % of data traffic. In contrast to that, 58 % of files have a small size (average 0.1 GB) make 8 % of dataset size and receive just 16 % of requests and 1 % of data traffic. The further analysis of the STAR

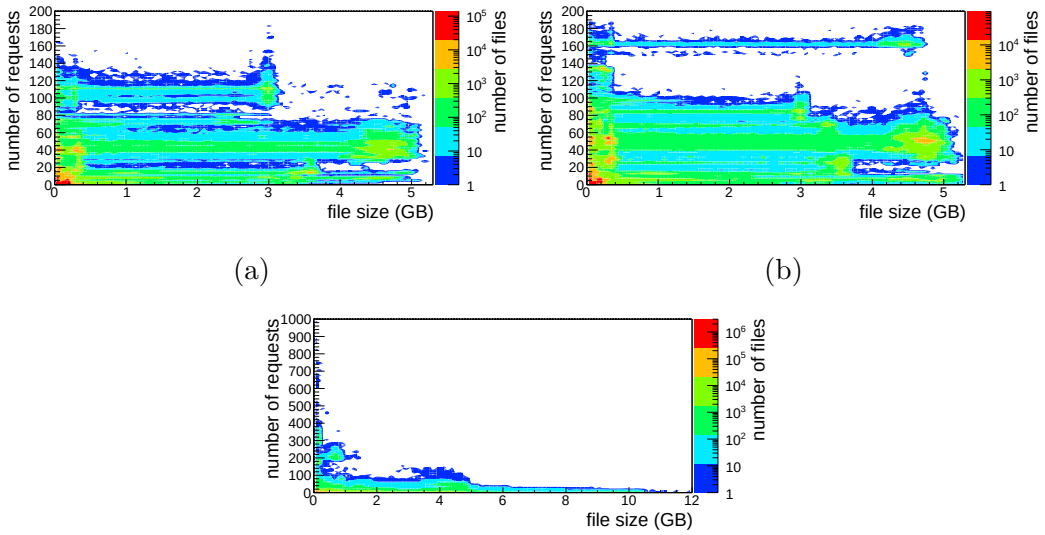


Figure 33: Data access patterns represented as contour-plots: (a) STAR1, (b) STAR2, (c) GOLIAS.

access patterns has shown that in both cases a set of approximately 6 % of files by total size can be selected in such a way, that it makes 20 % of requests and 18 % of network traffic. The horizontal lines on the plots reveal looping access pattern: repeated processing of a set of files of various size. The GOLIAS access pattern shows fewer features on the contour plot. It has two "tails" aside from the densest area: small files used hundreds of times and large files used tens of times. Keeping those small "popular" files in cache would obviously increase data access speed.

To summarize, the analysis of data access patterns has shown a heterogeneous distribution of files by size and access rates. The "popularity" of a file cannot be predicted by its size alone. At the same time, there exist a subset of files which make a significant fraction of requests and data flow. Therefore, an algorithm which would keep such files in cache can improve overall performance.

9.2 Summary of caching algorithms

The efficiency of caching can be estimated by two quantities, the cache hits H and cache hits per megabyte of data H_d(cache data hits):

$$H = \frac{N_{cache}}{N_{req} - N_{set}} \tag{30}$$

$$H_d = \frac{S_{cache}}{S_{req} - S_{set}} \tag{31}$$

where N_{req} is the total number of requests, S_{req} is the total amount of transferred data in bytes, N_{set} is the number of unique filenames, S_{set} is the total size of unique files in bytes, N_{cache} is the number of files transferred from cache is S_{cache} is the amount of data transferred from cache in bytes.

By maximizing the cache hits H one reduces the number of files transferred from external sources and thus reduces the overall makespan due to transfer startup overhead for each file. By maximizing the cache data hits H_d one reduces the network load and transmission overhead, since more data is reused from the local cache.

If the access pattern is completely random, the expected cache hit and cache data hits would be equal to *cache size/storage size*, so it can be useful to compare the actual cache performance to this estimation.

Altogether 27 different caching algorithms were simulated. But the majority of studied algorithms did not bring any improvements over the simplest one (FIFO). Only the algorithms that appeared to be the most efficient are discussed here:

- **First-In-First-Out (FIFO)**: evicts files in the same order they entered the cache. Performance of this trivial algorithm provides a good comparison benchmark against more sophisticated ones which can demand significant computational resources.

◯ **Least-Recently-Used (LRU)**: evicts the set of files which were not used for the longest period of time.

● **Least-Frequently-Used (LFU)**: evicts the set of files which were requested fewer times since they entered the cache.

★ **Most Size (MS)**: evicts the set of files which have the largest size.

+ **Adaptive Replacement Cache (ARC)** [217]: splits cached files into two lists: L1 contains files which were accessed once, and L2 contains files which were accessed more than ones since they entered the cache. LRU is then applied to both lists. The self adjustable parameter

$$a = \frac{cache\ hits\ in\ L1}{cache\ hits\ in\ L2} \tag{32}$$

defines the number of cached files in each list. The general idea is to invest more in the list which delivers more hits.

∗ **Least Value based on Caching Time (LVCT)** [218]: Deletes files with the smallest value of the Utility Function:

$$UtilityFunction = \frac{1}{CachingTime \cdot FileSize} \tag{33}$$

where $CachingTime$ of a file F is the total size of all files accessed after the last request for the file F.

▽ **Improved-Least Value based on Caching Time (ILVCT)** [219]: Deletes files with the smallest value of the Utility Function:

$$UtilityFunction = \frac{1}{AccessedFiles \cdot CachingTime \cdot FileSize} \tag{34}$$

where $CachingTime$ is the same as for LVCT and $AccessedFiles$ is a number of files requested after the last request for the selected file.

9.3 Evaluation and comparison of caching algorithms

Three series of simulations with three access patterns were performed for each algorithm (90 simulations in total for each algorithm):

- Ten simulations with cache size 1 – 90 % of the dataset with a fixed low mark 75% and a high-mark 95%. These simulations study the case when a large storage at a site is managed as cache. Such case is aligned with a DPM and Xrootd use where most (if not all) of the dataset resides on hard drive arrays.

- Ten simulations with cache size 1.2 – 0.0025% of the dataset with fixed low mark 75% and high-mark 85%. We used those simulations to understand the behavior of cache cleanup if the cache size is by several orders less than the dataset size. This is the case of a separate cache of data transfer tools or computing sites with a lack of storage space.

- Ten simulations with fixed cache size 10% of the dataset, fixed high-mark 95% and variable low mark within 0 – 90%. We performed those simulations to better understand the effect of data retention in cache (delete the least in hope of re-use).

In order to compare one algorithm against another an average improvement can be calculated in a following way:

$$Average\ improvement = \frac{\sum_{i=1}^{n} \frac{value2_i - value1_i}{value1_i}}{n}, \tag{35}$$

where n is the total amount of simulations with equal parameters for both algorithms, i is the number of the simulation, $value1_i$ is cache hits or cache data hits of a reference algorithm (FIFO) and $value2_i$ is cache hits or cache data hits of a compared algorithm.

Table 6 contains the results of comparison of all the algorithms presented

Table 6: Average improvement of caching algorithms over FIFO.

Algorithm	cache hits	cache data hits
MS	**116** %	-20 %
LRU	8 %	5 %
LFU	-27 %	-18 %
ARC	13 %	**11** %
LVCT	**86** %	2 %
ILVCT	28 %	2 %

here against FIFO. Results of simulation series 1 and 2 were used to calculate the average improvement (60 values for each algorithm). According to our results, the LFU algorithm does not bring any improvement over FIFO due to its well known flaw — it accumulates files which were popular for a short period of time, and those files prevent newer ones from staying in cache.

The ARC algorithm was developed as an improvement to LRU, and not surprisingly, it outperforms LRU by ≈5% in cache hits and ≈7% in cache data hits. Therefore, LFU and LRU algorithms could be excluded from the further analysis in our case studies.

The detailed results of simulations for all 3 series are given in Figures 34 – 36. The performance of FIFO and 3 algorithms appeared to be the most efficient (MS, ACR and LVCT) is presented at the plots. Difference between

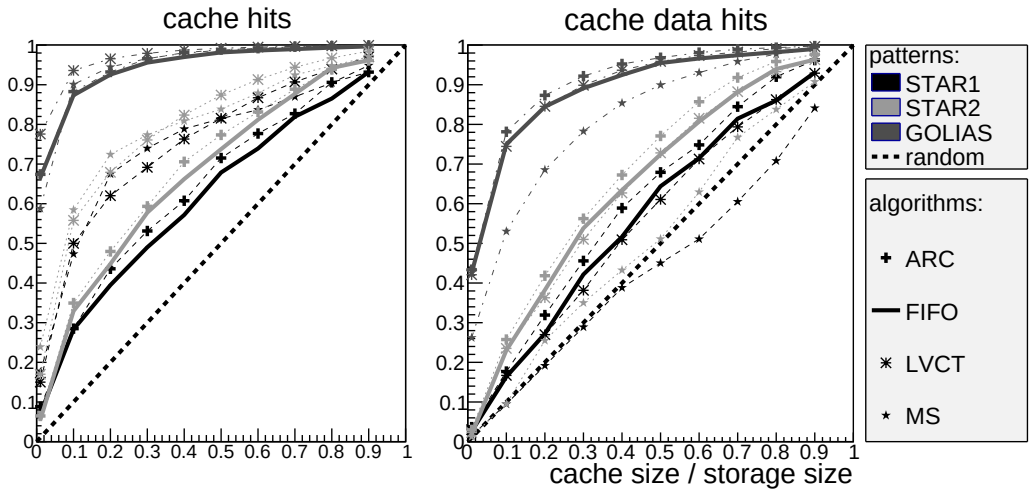


Figure 34: Simulated performance of caching algorithms for cache of large size.

Tier-2 and Tier-0 access patterns leads to distinct cache performance. Only the data dedicated for the ongoing analysis are placed at the Tier-2 site, while at the Tier-0 site all the experimental data are stored. As a result — the access pattern at the Tier-2 site has stronger access locality. STAR1 and STAR2 access patterns correspond to Tier-0 site and GOLIAS to a Tier-2 site. Thus, any particular algorithm at the plots delivers higher cache hits and cache data hits for GOLIAS access pattern than for STAR1 and STAR2.

The behavior of algorithms is similar within each dataset that is, their respective performance ordering is the same. This observation implies that if one of the algorithms appears to be the most efficient for one of the datasets it is also the most efficient for the other datasets. This statement is also true for the rest of simulated algorithms not present at our final figures. Though the communities represented by the STAR and GOLIAS access patterns are

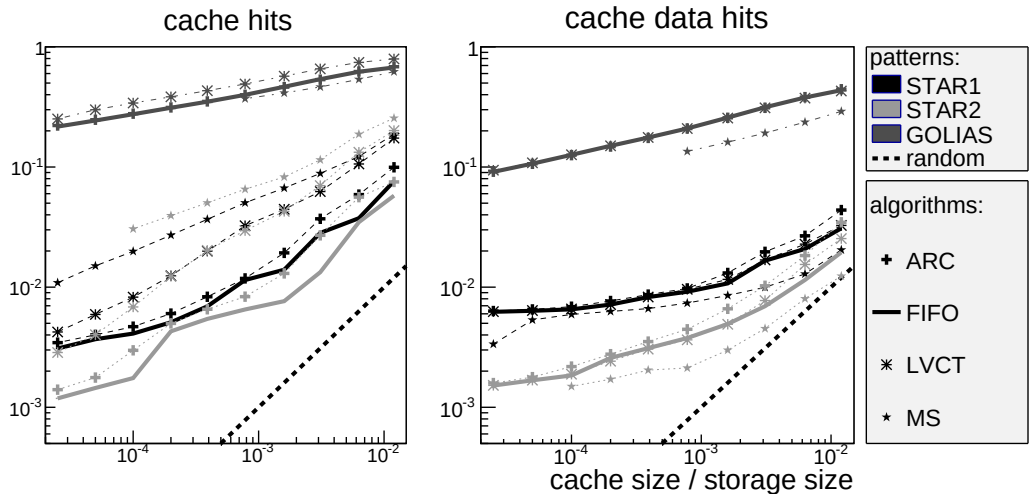


Figure 35: Simulated performance of caching algorithms for cache of small size.

somewhat similar, this result is slightly surprising as our case studies represent two time sequences within the same usage and totally uncorrelated experiments. It would be interesting to compare those algorithms in a different experimental context (outside the HENP communities) but such study is outside the scope of this work.

The MS algorithm has shown outstanding cache hits, but the lowest cache data hits. At the same time, the LVCT has cache hits comparable to the MS while cache data hits are 2% improved over the FIFO. This algorithm could be the preferable one when the cache hits is the target parameter for optimization. The ARC algorithm has shown the highest cache data hits for the studied access patterns.

The dependence of algorithms' performance on the low mark is presented in Figure 36. With higher low mark the number of clean-ups increases and that is why the difference between algorithms becomes more notable. Performance of efficient algorithms (FIFO, LRU, ARC and LVCT) increases steadily with the low mark. One should be careful when setting up a cache low mark at a particular site, since a higher low mark can increase cache performance significantly, but at the same time it can result in running cache clean-ups too often, consuming significant computational resources (and potentially increasing the chance to interfere with data transfers hence, degrad-

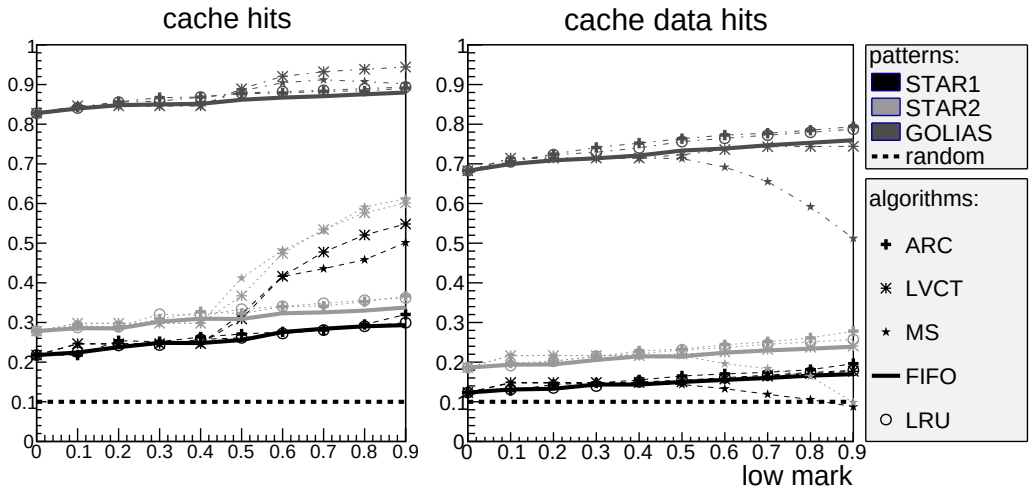


Figure 36: Simulated dependence of cache performance on the low mark.

ing transfer performances if delete/writes/read overlap).

Regardless of the cache size, Tier-level and specificity of an experiment the LVCT and ARC appeared to be the most efficient caching algorithms for the considered application. While we found the result surprising at first, we attribute this result to an access pattern which is intrinsically similar in nature. LVCT and ARC could certainly be safely used in data transfer and management tools for HENP computing.

- If the goal is to minimize makespan due to a transfer startup overhead the LVCT algorithm should be selected.
- If the goal is to minimize the network load and transmission overhead the ARC algorithm is an option.

10 Conclusion and outlook

10.1 Summary of the results

As the result of the Ph.D. thesis, a novel orchestration approach for large-scale distributed data-intensive computations is proposed. The approach introduces a new iterative scheduling method based on network flows algorithms with polynomial complexity. The scheduling problem at each iteration is formulated in such a way that its complexity depends on the number of sites and network links but not the number of jobs. As a consequence, the approach can effectively address processing of extensive datasets. A wide range of simulations using GridSim and data from real systems was performed for this research in order to demonstrate the viability of the approach on the rich set of problems. Work on this Ph.D. thesis was driven by the current computing needs of running High Energy and Nuclear Physics experiment — STAR, which offloads petabytes of data for remote processing. The results expand to similar HENP experiments, including the ones at CERN, as well as to other data-intensive fields.

Having carefully analyzed state of the art resource management in big data systems, we have concluded, that an important type of computations in HENP — data production, which accounts for a large fraction of all computations, is not fully addressed by the known scheduling approaches. Unlike other common big data workloads, data production cannot efficiently rely on spatial or temporal data locality, because there is no data reuse between the jobs. Offloading data production to remote sites, sometimes connected over unreliable networks, requires a different type of optimization with respect to job scheduling and data access than provided by common approaches. In practice, data production often involves ad-hoc setups and requires increased effort from the personnel. As the computing collaboration grows, aggregates more resources and, therefore, increases complexity and heterogeneity, more sophisticated orchestration of computing activities is needed for efficient operation. This work presents a solution to this problem and provides an automated, flexible and adaptable planning for data production.

We presented a formal description of the problem in a form of a constraint satisfaction problem. This model allowed to consider CPU allocation, data transfer and storage management within a single scheduling problem includ-

ing resource sharing. Previous known approaches (see Section 4) consider those interdependent problems separately (e.g. [191]) or partially ignore resource sharing (e.g. [141, 159]), which limits the possibilities for end-to-end optimization. We tested the model, analyzed its limitations and identified key factors for the primary use case. This allowed to develop a more specialized and efficient approach for data production.

The new approach is based on network flow algorithms. It includes necessary aspects for efficient orchestration of data production and at the same time remains computationally tractable. While a general job scheduling problem is known to be NP-hard, the proposed formulation allows to solve the considered case using a polynomial algorithm. We have implemented the planner and complemented it with the execution method, where a centrally created plan is processed by distributed services running at sites. Unlike workload management systems currently used, our approach prioritizes data placement and transfer over CPU allocation. It readjusts data distribution continuously and preemptively in order to match the observed system performance (network, CPU, storage) and maximize throughput of the computations.

Further, we have extended the base model to consider data provenance and replication across a distributed system. We also presented a method to optimize initial data distribution knowing the properties of upcoming computations and parameters of the system. Such initial data distribution combined with our planning allows to achieve peak system performance faster and further decrease the makespan. These extensions allow to apply our approach to a greater variety of systems, beyond specifics of the STAR experiment alone.

We have demonstrated efficiency of our planning approach in a wide scope of simulations based on data obtained from real systems. The simulations studied an influence of network performance on the overall efficiency of computations and compared our planner to other approaches used in practice. We have considered concurrent background traffic over shared networks, various network topologies, large-scale heterogeneous systems, non-uniform initial data distribution and presence of data replication. In all the studied use cases our planner has consistently demonstrated shorter makespan and better CPU utilization than other simulated approaches. For instance, in

randomly generated scale-free networks the improvement reaches 11 – 37 % compared to the approach currently used in practice. The measured short solving times confirm that the planner can be used in the HENP production environment and potentially in the largest prospective infrastructures.

We have also studied applicability of general caching policies to HENP data management. The evaluations revealed stability of data access patterns over time and across experiments. We have identified the policies which can be efficiently used along with our scheduling approach or independently for data management at sites.

While initially focused on distributed data production in HENP, our approach can also be helpful in other applications where a large set of data undergoes a single pass of processing on geographically spread resources. The approach is especially beneficial when the network performance is a limiting factor.

The results achieved within the work on this thesis were presented at the following international conferences: 2013/2014/2016 international workshop on Advanced Computing and Analysis Techniques in physics research (ACAT), 2016 international conference on Computing in High Energy and nuclear Physics (CHEP), 2016 IEEE Symposium Series on Computational Intelligence (IEEE SSCI), 2015 Multidisciplinary International Scheduling conference: Theory and Applications (MISTA). The corresponding contributions were published in Journal of Physics: Conference Series, Proceedings of the 7th Multidisciplinary International Scheduling Conference, Proceedings of the IEEE Symposium on Computational Intelligence in Scheduling and Network Design. A paper "Planning of distributed data production for High Energy and Nuclear Physics" is accepted for publication in the Cluster Computing journal.

10.2 Future work

Since our approach has been thoroughly tested in simulations based on real data and systems, the next planned step is to deploy it to the production environment of the STAR experiment. The deployment should include integration with other services, such as monitoring, workload management, data transfer tools and file catalog. Further, the planner can also benefit from integration with the components providing Software Defined Networking and/or

Dynamic Circuit Provisioning (see Section 4.3). This would allow to ensure bandwidth guarantees, controls over network streams according to instantiated plans, and proactively account for other concurrent network activities planned ahead. Such integration was envisioned during the design and testing of the planner. It does not require changes to our approach or external services but a proper information exchange between all the components.

The approach can also be adopted by other fields with workloads similar to data production in HENP. Such workloads feature data-intensive distributed computations with data-level parallelism, where each portion of data is processed once. It may include, for example, data cleansing, log analysis, feature extraction, event reconstruction from sensor data, image recognition and tagging, generation of subtitles for video files. We expect our approach to provide a considerable improvement in efficiency of computing for such workloads, which merits a future practical evaluation.

Further studies could also evaluate the optimization which our approach can possibly provide to another important type of computations in HENP — the simulations (see Section 5.1). Such computations do not require large volumes of input data, however, the amount of output data is comparable to data production. Here, we anticipate our approach to mitigate network and storage bottlenecks when transferring the output data from external sites to the permanent storage. The planner can also be helpful to predict how many simulation jobs can effectively run at each site given the network and storage status. For the embedding (event overlay) type of simulations, utilizing raw data, planning of input transfers may also become advantageous.

There are multiple possibilities to further extend our planning approach. First, it is possible to identify network links with high background traffic and decrease the load on those links using alternative transfer paths as illustrated in Figure 37. This should improve load balancing in worst-case scenarios of congested networks. Such reasoning can be implemented by monitoring recent network performance and setting higher costs to the network links where large background traffic is detected.

Second, multiple other practical optimizations can be achieved by adjustment of the cost function. For example, more reliable resources, faster CPUs and external clouds with lower lease price can be prioritized. Such extension requires to define the cost function in such a way that would balance the

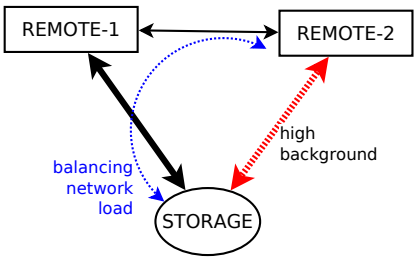


Figure 37: Load balancing in networks with high background traffic.

importance of the optimization goals. This allows fine-tuning of the planner according to secondary optimization goals, practical for real-life scenarios.

Third, optimization for energy efficiency can be naturally integrated into the proposed planning model. Analysis of instantiated plans can detect unused resources (e.g. machines and network hardware). Presence of idle resources in the plan indicates the existence of bottlenecks in the system which prevent efficient usage of those resources. Therefore, such resources can be powered off in order to reduce expenses. The proposed execution approach leaves enough time to bring those resources back by the time when their usage becomes efficient and is planned. Alternatively, resources not used for data production can be reassigned to other computing activities. For instance, if the network bandwidth is not sufficient to saturate all available CPUs at a remote site with raw data for data production, the unused CPUs can be assigned simulation jobs. Such automated reasoning added to our planner could greatly simplify resource provisioning typically performed by site and grid administrators.

As the ultimate goal, the planner could be extended to more general workloads, beyond the specifics of data production. The expected advantage is to have a uniform and efficient orchestration for many types of workflows within a large-scale computing collaboration/enterprise. This would require consideration of mutual dependency between the jobs and data re-usage, which could be based on our studies. Scheduling of execution, data placement and transfer considering each individual job was discussed in Section 6. Optimization of data transfer between N sources and M destinations was earlier addressed by Michal Zerola [11]. The further research may consider union the presented concepts into a multistage scheduling approach. In such approach

our planner can be used for efficient load balancing in the first scheduling stage. Then, at the further stages, data transfer path and CPU allocation for each job can be mapped onto the initial plan in order to provide fine-grained optimization. Such possibility deserves a careful study in future.

11 Bibliography

[1] Malte Schwarzkopf. The evolution of cluster scheduler architectures. Firmament Blog `http://firmament.io/blog/scheduler-architectures.html`. Accessed: September 2017.

[2] Tevfik Kosar and Miron Livny. Stork: Making data placement a first class citizen in the grid. In *24th International Conference on Distributed Computing Systems*, pages 342–349. IEEE, 2004.

[3] Jamie Shiers. The Worldwide LHC Computing Grid (Worldwide LCG). *Computer physics communications*, 177:219–223, 2007.

[4] Ian Bird et al. Update of the Computing Models of the WLCG and the LHC Experiments. Technical Report CERN-LHCC-2014-014. LCG-TDR-002, CERN, Geneva, 2014.

[5] Costin Caramarcu, Christopher Hollowell, William Strecker-Kellogg, Antonio Wong, and Alexandr Zaytsev. The role of dedicated data computing centers in the age of cloud computing. *Journal of Physics: Conference Series*, 898(8), 2017.

[6] Bruce Gibbard and Thomas Throwe. The RHIC computing facility. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 499(2):814 – 818, 2003.

[7] Ian Bird. LHC computing (WLCG): Past, present, and future. *Grid and Cloud Computing: Concepts and Practical Applications*, 192:1–29, 2016.

[8] HB Newman, A Mughal, I Kassymkhanova, J Bunn, R Voicu, V Lápádátescu, and D Kcira. Networking for high energy physics. *Grid and Cloud Computing: Concepts and Practical Applications*, 192, 2016.

[9] Artur Barczyk. Advanced networking for scientific applications. *Grid and Cloud Computing: Concepts and Practical Applications*, 192:185–205, 2016.

[10] Shishir Bharathi and Ann Chervenak. Data staging strategies and their impact on the execution of scientific workflows. In *2nd International workshop on data-aware distributed computing*. ACM, 2009.

[11] Michal Zerola, Jérôme Lauret, Roman Barták, and Michal Šumbera. One click dataset transfer: toward efficient coupling of distributed storage resources and CPUs. *Journal of Physics: Conference Series*, 368(1), 2012.

[12] Joanna Kołodziej and Samee Ullah Khan. Data scheduling in data grids and data centers: a short taxonomy of problems and intelligent resolution techniques. In *Transactions on computational collective intelligence X*, pages 103–119. Springer, 2013.

[13] Zhang Junwei, Lee Bu-Sung, Tang Xueyan, and Yeo Chai-Kiat. Impact of parallel download on job scheduling in data grid environment. In *7th International Conference on Grid and Cooperative Computing*, pages 102–109, 2008.

[14] Marco Cattaneo, Philippe Charpentier, Peter Clarke, and Stefan Roiser. Recent and planned changes to the LHCb computing model. *Journal of Physics: Conference Series*, 513(3), 2014.

[15] Dagmar Adamova, Jiri Chudoba, Marek Elias, Lukas Fiala, Tomas Kouba, Milos Lokajicek, and Jan Svec. WLCG Tier-2 site in Prague: a little bit of history, current status and future perspectives. *Journal of Physics: Conference Series*, 608, 2015.

[16] Jiří Horký, Miloš Lokajíček, and Jakub Peisar. Influence of distributing a Tier-2 data storage on physics analysis. 15th Int. Workshop on Advanced Computing and Analysis Techniques in Physical Research (2013) Accessed: December 2013.

[17] Latchezar Betev, Andrei Gheata, Mihaela Gheata, Costin Grigoras, and Peter Hristov. Performance optimisations for distributed analysis in ALICE. *Journal of Physics: Conference Series*, 523(1), 2014.

[18] Han Hu, Yonggang Wen, Tat-Seng Chua, and Xuelong Li. Toward scalable systems for big data analytics: A technology tutorial. *IEEE Access*, 2:652–687, 2014.

[19] Geydar Agakishiev, Nikita Balashov, Wayne Betts, Lidia Didenko, Levente Hajdu, Vladimir Korenkov, Evgeniy Kuznetsov, Jérôme Lauret, Valery Mitsyn, and Yury Panebratsev. STAR's approach to highly efficient end-to-end grid production. In *26th International Symposium on Nuclear Electronics and Computing*, 2017.

[20] K. H. Ackermann et al. STAR detector overview. *Nuclear Instruments and Methods in Physics Research*, A499:624–632, 2003.

[21] Levente Hajdu et al. STAR experience with automated high efficiency Grid based data production framework at KISTI/Korea. In *HEPiX Workshop*. Oxford University, UK, 2015.

[22] Jan Balewski, Jerome Lauret, Doug Olson, Iwona Sakrejda, Dmitry Arkhipkin, et al. Offloading peak processing to virtual farm by STAR experiment at RHIC. *Journal of Physics: Conference Series*, 368(012011), 2012.

[23] Dzmitry Makatun, Jérôme Lauret, Hana Rudová, and Michal Šumbera. Planning for distributed workflows: constraint-based coscheduling of computational jobs and data placement in distributed environments. *Journal of Physics: Conference Series*, 608(1), 2015.

[24] Dzmitry Makatun, Jérôme Lauret, Hana Rudová, and Michal Šumbera. Model for planning of distributed data production. In *Proceedings of the 7th Multidisciplinary International Scheduling Conference (MISTA)*, pages 699–703, 2015.

[25] Dzmitry Makatun, Jérôme Lauret, and Hana Rudová. Planning of distributed data production for High Energy and Nuclear Physics. *Cluster Computing*, 2018. (Accepted).

[26] Dzmitry Makatun, Jérôme Lauret, Hana Rudová, and Michal Šumbera. Simulations and study of a new scheduling approach for distributed data production. *Journal of Physics: Conference Series*, 762(1), 2016.

[27] Dzmitry Makatun, Jérôme Lauret, Hana Rudová, and Michal Šumbera. Network flows for data distribution and computation. *Proceedings of the IEEE Symposium on Computational Intelligence in Scheduling and Network Design*, 2016.

[28] Dzmitry Makatun, Jérôme Lauret, Hana Rudová, and Michal Šumbera. Provenance-aware optimization of workload for distributed data production. *Journal of Physics: Conference Series*, 898(5), 2017.

[29] Dzmitry Makatun, Jérôme Lauret, and Michal Šumbera. Study of cache performance in distributed environment for data processing. *Journal of Physics: Conference Series*, 523(1), 2014.

[30] Doug Laney. 3D data management: Controlling data volume, velocity and variety. *META Group Research Note*, 6(70), 2001.

[31] John Gantz and David Reinsel. Extracting value from chaos. *IDC iView*, 1142, 2011.

[32] Michael Cooper and Peter Mell. Tackling big data, NIST information technology laboratory, computer security division. In *Federal Computer Security Managers' Forum*, 2012.

[33] Mike Loukides. *What is data science?* O'Reilly Media, Inc., 2011.

[34] CL Philip Chen and Chun-Yang Zhang. Data-intensive applications, challenges, techniques and technologies: A survey on big data. *Information Sciences*, 275:314–347, 2014.

[35] Min Chen, Shiwen Mao, and Yunhao Liu. Big data: A survey. *Mobile Networks and Applications*, 19:171–209, 2014.

[36] Raghavendra Kune, Pramod Kumar Konugurthi, Arun Agarwal, Raghavendra Rao Chillarige, and Rajkumar Buyya. The anatomy of big data computing. *Software: Practice and Experience*, 46:79–105, 2016.

[37] Mayer-Schönberger Viktor and Cukier Kenneth. *Big data: A revolution that will transform how we live, work, and think*. 2013.

[38] Ahmed Abbasi, Suprateek Sarker, and Roger HL Chiang. Big data research in information systems: Toward an inclusive research agenda. *Journal of the Association for Information Systems*, 17(2), 2016.

[39] Vivien Marx. Biology: The big challenges of big data. *Nature*, 498:255–260, 2013.

[40] Tony Hey, Stewart Tansley, Kristin M Tolle, et al. *The fourth paradigm: data-intensive scientific discovery*, volume 1. Microsoft research Redmond, WA, 2009.

[41] Trudie Lang. Advancing global health research through digital technology and sharing data. *Science*, 331:714–717, 2011.

[42] Jonathan T Overpeck, Gerald A Meehl, Sandrine Bony, and David R Easterling. Climate data challenges in the 21st century. *Science*, 331:700–702, 2011.

[43] Gary King. Ensuring the data-rich future of the social sciences. *Science*, 331:719–721, 2011.

[44] Ewa Deelman et al. GriPhyN and LIGO, building a virtual data grid for gravitational wave scientists. In *11th IEEE International Symposium on High Performance Distributed Computing, HPDC-11*, pages 225–234. IEEE, 2002.

[45] David P Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. SETI@ home: an experiment in public-resource computing. *Communications of the ACM*, 45:56–61, 2002.

[46] Rupak Biswas. NASA advanced computing environment for science and engineering. `https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20170007408.pdf`. Accessed: December 2017.

[47] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 51:107–113, 2008.

[48] Doug Beaver, Sanjeev Kumar, Harry C Li, Jason Sobel, Peter Vajgel, et al. Finding a needle in Haystack: Facebook's photo storage. In *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, volume 10, 2010.

[49] Roshan Sumbaly, Jay Kreps, and Sam Shah. The big data ecosystem at LinkedIn. In *ACM SIGMOD International Conference on Management of Data*, pages 1125–1134. ACM, 2013.

[50] Amir Gandomi and Murtaza Haider. Beyond the hype: Big data concepts, methods, and analytics. *International Journal of Information Management*, 35(2):137 – 144, 2015.

[51] Seref Sagiroglu and Duygu Sinanc. Big data: A review. In *International Conference on Collaboration Technologies and Systems (CTS)*, pages 42–47. IEEE, 2013.

[52] Sunil Erevelles, Nobuyuki Fukawa, and Linda Swayne. Big data consumer analytics and the transformation of marketing. *Journal of Business Research*, 69(2):897 – 904, 2016.

[53] Matthew A. Waller and Stanley E. Fawcett. Data science, predictive analytics, and big data: A revolution that will transform supply chain design and management. *Journal of Business Logistics*, 34(2):77–84, 2013.

[54] Shantenu Jha, Judy Qiu, Andre Luckow, Pradeep Mantha, and Geoffrey C Fox. A tale of two data-intensive paradigms: Applications, abstractions, and architectures. In *IEEE International Congress on Big Data*, pages 645–652, 2014.

[55] Hamid Reza Asaadi, Dounia Khaldi, and Barbara Chapman. A comparative survey of the HPC and big data paradigms: Analysis and experiments. In *IEEE International Conference on Cluster Computing (CLUSTER)*, pages 423–432, 2016.

[56] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM Sigact News*, 33:51–59, 2002.

[57] Theo Haerder and Andreas Reuter. Principles of transaction-oriented database recovery. *ACM Computing Surveys (CSUR)*, 15:287–317, 1983.

[58] Dan Pritchett. BASE: An ACID alternative. *Queue*, 6:48–55, 2008.

[59] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google file system. In *SIGOPS Operating Systems Review*, volume 37, pages 29–43. ACM, 2003.

[60] Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for MapReduce. In *21st annual ACM-SIAM symposium on Discrete Algorithms*, pages 938–948, 2010.

[61] Tom White. *Hadoop: The definitive guide*. O'Reilly Media, Inc., 2012.

[62] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The Hadoop Distributed File System. In *IEEE 26th Symposium on mass storage systems and technologies*. IEEE, 2010.

[63] Spencer Shepler, Mike Eisler, David Robinson, Brent Callaghan, Robert Thurlow, David Noveck, and Carl Beame. Network file system (NFS) version 4 protocol. `https://tools.ietf.org/html/rfc3530`, 2003. Accessed: January 2018.

[64] John H Howard et al. *An overview of the Andrew File System*. Carnegie Mellon University, Information Technology Center, 1988.

[65] Xrootd. `http://xrootd.slac.stanford.edu/`. Accessed: September 2017.

[66] Jakob Blomer, Carlos Aguado-Sánchez, Predrag Buncic, and Artem Harutyunyan. Distributing LHC application software and conditions databases using the CernVM file system. *Journal of Physics: Conference Series*, 331(4), 2011.

[67] Philip Schwan et al. Lustre: Building a file system for 1000-node clusters. In *Proceedings of the Linux symposium*, pages 380–386, 2003.

[68] Sage A Weil, Scott A Brandt, Ethan L Miller, Darrell DE Long, and Carlos Maltzahn. Ceph: A scalable, high-performance distributed file system. In *7th symposium on Operating systems design and implementation*, pages 307–320. USENIX, 2006.

[69] Michael Poat, Jérôme Lauret, and Wayne Betts. POSIX and object distributed storage systems performance comparison studies with real-life scenarios in an experimental data taking context leveraging OpenStack Swift and Ceph. *Journal of Physics: Conference Series*, 664, 2015.

[70] Michael Poat and Jérôme Lauret. Performance and advanced data placement techniques with Ceph's distributed storage system. *Journal of Physics: Conference Series*, 762, 2016.

[71] Alex Davies and Alessandro Orsaria. *Linux Journal*, 2013(235), 2013.

[72] Jakob Blomer. Experiences on file systems: Which is the best file system for you? *Journal of Physics: Conference Series*, 664(4), 2015.

[73] Yuduo Zhou. Large scale distributed file system survey. `http://grids.ucs.indiana.edu/ptliupages/publications/Large%20Scale%20Distributed%20File%20System%20Survey.pdf`. Accessed: September 2017.

[74] Maria Girone. Distributed data management and distributed file systems. *Journal of Physics: Conference Series*, 664(4), 2015.

[75] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *ACM SIGOPS operating systems review*, volume 41, pages 59–72. ACM, 2007.

[76] Marc Snir. *MPI–the Complete Reference: the MPI core*, volume 1. MIT press, 1998.

[77] Barbara Chapman, Gabriele Jost, and Ruud Van Der Pas. *Using OpenMP: portable shared memory parallel programming*, volume 10. MIT press, 2008.

[78] George Almasi. PGAS (Partitioned Global Address Space) languages. In *Encyclopedia of Parallel Computing*, pages 1539–1545. Springer, 2011.

[79] Adaptive Computing and Green Computing. Torque resource manager. `http://www.adaptivecomputing.com`. Accessed: November 2015.

[80] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: the Condor experience. *Concurrency and Computation: Practice and Experience*, 17:323–356, 2005.

[81] Stuart K Paterson and Andrei Tsaregorodtsev. DIRAC optimized workload management. *Journal of Physics: Conference Series*, 119(6), 2008.

[82] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10, 2010.

[83] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, et al. Apache Spark: A unified engine for big data processing. *Communications of the ACM*, 59:56–65, 2016.

[84] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012.

[85] Shanjiang Tang, Bingsheng He, Haikun Liu, and Bu-Sung Lee. Resource management in big data processing systems. `http://www.comp.nus.edu.sg/~hebs/pub/shangresource16.pdf`. Accessed: March 2017.

[86] Jay Kreps, Neha Narkhede, Jun Rao, et al. Kafka: A distributed messaging system for log processing. In *6th International Workshop on Networking Meets Databases (NetDB 2011)*, 2011.

[87] Rajiv Ranjan. Streaming big data processing in datacenter clouds. *IEEE Cloud Computing*, 1:78–83, 2014.

[88] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26, 2008.

[89] Lars George. *HBase: the definitive guide: random access to your planet-size data*. O'Reilly Media, Inc., 2011.

[90] Kristina Chodorow. *MongoDB: The Definitive Guide: Powerful and Scalable Data Storage*. O'Reilly Media, Inc., 2013.

[91] Avinash Lakshman and Prashant Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44:35–40, 2010.

[92] Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, et al. Apache Hadoop YARN: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, 2013.

[93] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D Joseph, Randy H Katz, Scott Shenker, and Ion Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation*, volume 11, 2011.

[94] Charlie Catlett, William E Allcock, Phil Andrews, Ruth Aydt, Ray Bair, Natasha Balac, Bryan Banister, Trish Barker, Mark Bartelt, Pete

Beckman, et al. Teragrid: Analysis of organization, system architecture, and middleware enabling new types of applications. Technical report, IOS Press, 2008.

[95] Ruth Pordes, Don Petravick, Bill Kramer, Doug Olson, Miron Livny, Alain Roy, Paul Avery, Kent Blackburn, Torre Wenaus, Frank Würthwein, et al. The Open Science Grid. *Journal of Physics: Conference Series*, 78(1), 2007.

[96] European Grid Infrastructure (EGI). `https://www.egi.eu/`. Accessed: June 2017.

[97] Ian Foster and Carl Kesselman. The history of the grid. *Advances in Parallel Computing*, 20:3–30, 2010.

[98] Šimon Tóth and Miroslav Ruda. Distributed job scheduling in MetaCentrum. *Journal of Physics: Conference Series*, 608(1), 2015.

[99] Peter Mell, Tim Grance, et al. The NIST definition of cloud computing. computer security division, information technology laboratory, national institute of standards and technology gaithersburg. `http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf`, 2011. November: June 2017.

[100] Seyyed Mohsen Hashemi and Amid Khatibi Bardsiri. Cloud computing vs. grid computing. *ARPN journal of systems and software*, 2:188–194, 2012.

[101] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop*. IEEE, 2008.

[102] Raquel V Lopes and Daniel Menascé. A taxonomy of job scheduling on distributed computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 27:3412–3428, 2016.

[103] B Bockelman et al. Commissioning the HTCondor-CE for the Open Science Grid. *Journal of Physics: Conference Series*, 664(6), 2015.

[104] Malte Schwarzkopf, Andy Konwinski, Michael Abd-El-Malek, and John Wilkes. Omega: flexible, scalable schedulers for large compute clusters. In *8th ACM European Conference on Computer Systems*, pages 351–364. ACM, 2013.

[105] Eric Boutin, Jaliya Ekanayake, Wei Lin, Bing Shi, Jingren Zhou, Zhengping Qian, Ming Wu, and Lidong Zhou. Apollo: Scalable and coordinated scheduling for cloud-scale computing. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, volume 14, pages 285–300, 2014.

[106] Nomad. `https://www.nomadproject.io/docs/internals/scheduling.html`. Accessed: September 2017.

[107] Kay Ousterhout, Patrick Wendell, Matei Zaharia, and Ion Stoica. Sparrow: Distributed, low latency scheduling. In *24th ACM Symposium on Operating Systems Principles*, pages 69–84, 2013.

[108] Christina Delimitrou, Daniel Sanchez, and Christos Kozyrakis. Tarcil: Reconciling scheduling speed and quality in large shared clusters. In *Proceedings of the 6th ACM Symposium on Cloud Computing*, pages 97–110, 2015.

[109] Konstantinos Karanasos, Sriram Rao, Carlo Curino, Chris Douglas, Kishore Chaliparambil, Giovanni Matteo Fumarola, Solom Heddaya, Raghu Ramakrishnan, and Sarvesh Sakalanaga. Mercury: Hybrid centralized and distributed scheduling in large shared clusters. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pages 485–497, 2015.

[110] Pamela Delgado, Florin Dinu, Anne-Marie Kermarrec, and Willy Zwaenepoel. Hawk: Hybrid datacenter scheduling. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pages 499–510, 2015.

[111] Joseph Skovira, Waiman Chan, Honbo Zhou, and David Lifka. The EASY—LoadLeveler API project. In *Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science*, volume 1162, pages 41–47. Springer, 1996.

[112] Songnian Zhou. LSF: Load sharing in large heterogeneous distributed systems. In *1st Workshop on cluster computing*, volume 136, 1992.

[113] David Jackson, Quinn Snell, and Mark Clement. Core algorithms of the Maui scheduler. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 87–102. Springer, 2001.

[114] Cray Inc. Introducing NQE. Technical Report 2153 2.97, Seattle, WA, 1997.

[115] Brett Bode, David M Halstead, Ricky Kendall, Zhou Lei, and David Jackson. The Portable Batch Scheduler and the Maui scheduler on Linux clusters. In *Annual Linux Showcase and Conference*, 2000.

[116] Wesley Emeneker, Dave Jackson, Joshua Butikofer, and Dan Stanzione. Dynamic virtual clustering with Xen and Moab. In *International Symposium on Parallel and Distributed Processing and Applications*, pages 440–451. Springer, 2006.

[117] Ian Foster. The globus toolkit for grid computing. In *Cluster Computing and the Grid*, volume 1. IEEE, 2001.

[118] Anand Natrajan, Marty A Humphrey, and Andrew S Grimshaw. Grid resource management in Legion. In *Grid resource management*, pages 145–160. Springer, 2004.

[119] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at Google with Borg. In *10th European Conference on Computer Systems*. ACM, 2015.

[120] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. Borg, Omega, and Kubernetes. *Communications of the ACM*, 59:50–57, 2016.

[121] Ionel Gog, Malte Schwarzkopf, Adam Gleave, Robert N. M. Watson, and Steven Hand. Firmament: Fast, centralized cluster scheduling at scale. In *12th USENIX Conference on Operating Systems Design and Implementation*, pages 99–115. USENIX Association, 2016.

[122] Muhammad Bilal Qureshi, Maryam Mehri Dehnavi, Nasro Min-Allah, Muhammad Shuaib Qureshi, Hameed Hussain, Ilias Rentifis, Nikos Tziritas, Thanasis Loukopoulos, Samee U Khan, Cheng-Zhong Xu, et al. Survey on grid resource allocation mechanisms. *Journal of Grid Computing*, 12:399–441, 2014.

[123] Fatos Xhafa and Ajith Abraham. *Metaheuristics for scheduling in distributed computing environments*, volume 146. Springer, 2008.

[124] Srikumar Venugopal, Rajkumar Buyya, and Kotagiri Ramamohanarao. A taxonomy of data grids for distributed data sharing, management, and processing. *ACM Computing Surveys (CSUR)*, 38(1):B1–B53, 2006.

[125] Jia Yu and Rajkumar Buyya. A taxonomy of workflow management systems for grid computing. *Journal of Grid Computing*, 3:171–200, 2005.

[126] Maria Alejandra Rodriguez and Rajkumar Buyya. A taxonomy and survey on scheduling algorithms for scientific workflows in IaaS cloud computing environments. *Concurrency and Computation: Practice and Experience*, 29(8), 2017.

[127] C Vijaya and DRP Srinivasan. A survey on resource scheduling in cloud computing. *International Journal of Pharmacy and Technology (IJPT)*, 8:26142–26162, 2016.

[128] Sucha Smanchat and Kanchana Viriyapant. Taxonomies of workflow scheduling problem and techniques in the cloud. *Future Generation Computer Systems*, 52, 2015.

[129] Yoav Etsion and Dan Tsafrir. A short survey of commercial cluster batch schedulers. *School of Computer Science and Engineering, The Hebrew University of Jerusalem*, 44221, 2005.

[130] A Forti, A Pérez-Calero Yzquierdo, T Hartmann, M Alef, A Lahiff, J Templon, S Dal Pra, M Gila, S Skipsey, C Acosta-Silva, et al. Multicore job scheduling in the Worldwide LHC Computing Grid. *Journal of Physics: Conference Series*, 664(6), 2015.

[131] Fatos Xhafa and Ajith Abraham. Computational models and heuristic methods for Grid scheduling problems. *Future generation computer systems*, 26(4):608–621, 2010.

[132] Dalibor Klusáček and Hana Rudová. Multi-resource aware fairsharing for heterogeneous systems. In *Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science*, number 8828, pages 53–69. Springer, 2014.

[133] Dalibor Klusáček and Hana Rudová. A metaheuristic for optimizing the performance and the fairness in job scheduling systems. In *Artificial Intelligence Applications in Information and Communication Technologies*, volume 607, pages 3–29. Springer, 2015.

[134] Srividya Srinivasan, Rajkumar Kettimuthu, Vijay Subramani, and Ponnuswamy Sadayappan. Selective reservation strategies for backfill job scheduling. In *Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science*, volume 2537, pages 55–71. Springer, 2002.

[135] Fatos Xhafa, Javier Carretero, Leonard Barolli, and Arjan Durresi. Immediate mode scheduling in grid systems. *International Journal of Web and Grid Services*, 3(2):219–236, 2007.

[136] Fatos Xhafa, Leonard Barolli, and Arjan Durresi. Batch mode scheduling in grid systems. *International Journal of Web and Grid Services*, 3(1):19–37, 2007.

[137] David A Lifka. The ANL/IBM SP scheduling system. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 295–303. Springer, 1995.

[138] Ahuva W. Mu'alem and Dror G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Transactions on Parallel and Distributed Systems*, 12:529–543, 2001.

[139] David Talby and Dror G Feitelson. Supporting priorities and improving utilization of the IBM SP scheduler using slack-based backfilling.

In *Symposium on Parallel and Distributed Processing*, pages 513–517. IEEE, 1999.

[140] Dalibor Klusáček and Hana Rudová. A metaheuristic for optimizing the performance and the fairness in job scheduling systems. In *Artificial Intelligence Applications in Information and Communication Technologies*, pages 3–29. Springer, 2015.

[141] Henri Casanova, Arnaud Legrand, Dmitrii Zagorodnov, and Francine Berman. Heuristics for scheduling parameter sweep applications in grid environments. In *Proceedings of the 9th Heterogeneous Computing Workshop*, pages 349–363. IEEE, 2000.

[142] Holger H Hoos and Thomas Stützle. *Stochastic local search: Foundations and applications*. Elsevier, 2004.

[143] Xingwu Zheng, Zhou Zhou, Xu Yang, Zhiling Lan, and Jia Wang. Exploring plan-based scheduling for large-scale computing systems. In *International Conference on Cluster Computing (CLUSTER)*, pages 259–268. IEEE, 2016.

[144] Dalibor Klusáček and Hana Rudová. Performance and fairness for users in parallel job scheduling. In *Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science*, number 7698, pages 235–252. Springer, 2013.

[145] Javier Carretero, Fatos Xhafa, and Ajith Abraham. Genetic algorithm based schedulers for grid computing systems. *International Journal of Innovative Computing, Information and Control*, 3(6):1–19, 2007.

[146] Pablo Moscato et al. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech concurrent computation program, C3P Report*, 826, 1989.

[147] Graham Ritchie. Static multi-processor scheduling with ant colony optimisation & local search, 2003.

[148] James Kennedy. Particle swarm optimization. In *Encyclopedia of machine learning*, pages 760–766. Springer, 2011.

[149] Hongbo Liu, Ajith Abraham, and Aboul Ella Hassanien. Scheduling jobs on computational grids using a fuzzy particle swarm optimization algorithm. *Future Generation Computer Systems*, 26(8):1336 – 1343, 2010.

[150] Christina Delimitrou and Christos Kozyrakis. Paragon: QoS-aware scheduling for heterogeneous datacenters. In *ACM SIGPLAN Notices*, volume 48, pages 77–88. ACM, 2013.

[151] Christina Delimitrou and Christos Kozyrakis. Quasar: resource-efficient and QoS-aware cluster management. In *ACM SIGPLAN Notices*, volume 49, pages 127–144. ACM, 2014.

[152] Aram Galstyan, Karl Czajkowski, and Kristina Lerman. Resource allocation in the grid using reinforcement learning. In *International Joint Conference on Autonomous Agents and Multiagent Systems*, volume 3, pages 1314–1315. IEEE, 2004.

[153] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. *Network flows: theory, algorithms, and applications*. Prentice Hall, 1993.

[154] Michael Isard, Vijayan Prabhakaran, Jon Currey, Udi Wieder, Kunal Talwar, and Andrew Goldberg. Quincy: fair scheduling for distributed computing clusters. In *SIGOPS 22nd Symposium on Operating Systems Principles*, pages 261–276. ACM, 2009.

[155] David Bernstein. Containers and cloud: From lxc to Docker to Kubernetes. *IEEE Cloud Computing*, 1:81–84, 2014.

[156] Michael R Garey and David S Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. W.H. Freeman and Co, New York, 1979.

[157] Maciej Drozdowski. *Scheduling for Parallel Processing*. Springer, 1st edition, 2009.

[158] Oliver Sinnen. *Task Scheduling for Parallel Systems*. Wiley Series on Parallel and Distributed Computing. Wiley, 2007.

[159] Elizeu Santos-Neto, Walfredo Cirne, Francisco Brasileiro, and Aliandro Lima. Exploiting replication and data reuse to efficiently schedule data-intensive applications on grids. In *Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science*, volume 3277, pages 210–232, 2005.

[160] Marcio Faerman, Alan Su, Richard Wolski, and Francine Berman. Adaptive performance prediction for distributed data-intensive applications. In *ACM/IEEE Conference on Supercomputing*, pages 36–51.

[161] Dzmitry Kliazovich, Pascal Bouvry, and Samee Ullah Khan. DENS: data center energy-efficient network-aware scheduling. *Cluster computing*, 16:65–75, 2013.

[162] Thomas Phan, Kavitha Ranganathan, and Radu Sion. Evolving toward the perfect schedule: Co-scheduling job assignments and data replication in wide-area systems using a genetic algorithm. In *Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science*, volume 3834, pages 173–193. Springer, 2005.

[163] Olivier Beaumont, Larry Carter, Jeanne Ferrante, Arnaud Legrand, and Yves Robert. Bandwidth-centric allocation of independent tasks on heterogeneous platforms. In *Parallel and Distributed Processing Symposium*. IEEE, 2001.

[164] R Kingsy Grace and R Manimegalai. Dynamic replica placement and selection strategies in data grids—a comprehensive survey. *Journal of Parallel and Distributed Computing*, 74:2099–2108, 2014.

[165] Uras Tos, Riad Mokadem, Abdelkader Hameurlain, Tolga Ayav, and Sebnem Bora. Dynamic replication strategies in data grid systems: a survey. *The Journal of Supercomputing*, 71:4116–4140, 2015.

[166] Vincent Garonne, Graeme A Stewart, Mario Lassnig, Angelos Molfetas, Martin Barisits, Thomas Beermann, Armin Nairz, Luc Goossens, Fernando Barreiro Megino, Cedric Serfon, et al. The ATLAS distributed data management project: Past and future. *Journal of Physics: Conference Series*, 396(3), 2012.

[167] Douglas Thain, Jim Basney, Se-Chang Son, and Miron Livny. The Kangaroo approach to data movement on the grid. In *10th IEEE International Symposium on High Performance Distributed Computing*, pages 325–333. IEEE, 2001.

[168] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A Lee, Jing Tao, and Yang Zhao. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, 2006.

[169] Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip J Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira da Silva, Miron Livny, et al. Pegasus, a workflow management system for science automation. *Future Generation Computer Systems*, 46:17–35, 2015.

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

[171] Vincent Garonne, R Vigne, G Stewart, M Barisits, M Lassnig, C Serfon, L Goossens, A Nairz, et al. Rucio–the next generation of large scale distributed system for ATLAS data management. *Journal of Physics: Conference Series*, 513(4), 2014.

[172] Ann L Chervenak, Alex Sim, Junmin Gu, Robert Schuler, and Nandan Hirpathak. Efficient data staging using performance-based adaptation and policy-based resource allocation. In *Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, pages 244–247. IEEE, 2014.

[173] Ann L Chervenak, Alex Sim, Junmin Gu, Robert E Schuler, and Nandan Hirpathak. Adaptation and policy-based resource allocation for efficient bulk data transfers in high performance computing environments. In *4th International Workshop on Network-Aware Data Management*. IEEE Press, 2014.

[174] Hasan Abbasi, Matthew Wolf, Greg Eisenhauer, Scott Klasky, Karsten Schwan, and Fang Zheng. Datastager: scalable data staging services for petascale applications. *Cluster Computing*, 13:277–290, 2010.

[175] Vlad Lapadatescu, Andrew Melo, Azher Mughal, Harvey Newman, Artur Barczyk, Paul Sheldon, Ramiro Voicu, Tony Wildish, Kaushik De, Iosif Legrand, et al. Integrating Network-Awareness and Network-Management into PhEDEx. *Proceedings of Science*, 2016.

[176] Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103:14–76, 2015.

[177] Bruno Astuto A Nunes, Marc Mendonca, Xuan-Nam Nguyen, Katia Obraczka, and Thierry Turletti. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys & Tutorials*, 16:1617–1634, 2014.

[178] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *7th USENIX Conference on Networked Systems Design and Implementation*, volume 10, 2010.

[179] Michael Bredel, Zdravko Bozakov, Artur Barczyk, and Harvey Newman. Flow-based load balancing in multipathed layer-2 networks using OpenFlow and multipath-TCP. In *3rd workshop on hot topics in software defined networking*, pages 213–214. ACM, 2014.

[180] Ian Foster, Alain Roy, and Volker Sander. A quality of service architecture that combines resource reservation and application adaptation. In *8th International Workshop on Quality of Service*, pages 181–188. IEEE, 2000.

[181] Iosif Legrand. Monitoring and control of large-scale distributed systems. *Grid and Cloud Computing: Concepts and Practical Applications*, 192, 2016.

[182] Di Xie, Ning Ding, Y Charlie Hu, and Ramana Kompella. The only constant is change: incorporating time-varying network reservations in data centers. *ACM SIGCOMM Computer Communication Review*, 42:199–210, 2012.

[183] Stefano Bagnasco et al. AliEn: ALICE environment on the grid. *Journal of Physics: Conference Series*, 119(6), 2008.

[184] MonAlisa: Grid online monitoring data of the ALICE experiment. `http://alimonitor.cern.ch/`. Accessed: October 2015.

[185] Kaushik De et al. The future of PanDA in ATLAS distributed computing. *Journal of Physics: Conference Series*, 664, 2015.

[186] Bengt Ahlgren, Christian Dannewitz, Claudio Imbrenda, Dirk Kutscher, and Borje Ohlman. A survey of information-centric networking. *IEEE Communications Magazine*, 50(7), 2012.

[187] George Xylomenos, Christopher N Ververidis, Vasilios A Siris, Nikos Fotiou, Christos Tsilopoulos, Xenofon Vasilakos, Konstantinos V Katsaros, and George C Polyzos. A survey of information-centric networking research. *IEEE Communications Surveys & Tutorials*, 16:1024–1049, 2014.

[188] Xueyan Tang and Jianliang Xu. QoS-aware replica placement for content distribution. *IEEE Transactions on parallel and distributed systems*, 16:921–932, 2005.

[189] Riad Mokadem and Abdelkader Hameurlain. Data replication strategies with performance objective in data grid systems: a survey. *International Journal of Grid and Utility Computing*, 6:30–46, 2014.

[190] Tarek Hamrouni, Sarra Slimani, and F Ben Charrada. A survey of dynamic replication and replica selection strategies based on data mining techniques in data grids. *Engineering Applications of Artificial Intelligence*, 48:140–158, 2016.

[191] Kavitha Ranganathan and Ian Foster. Decoupling computation and data scheduling in distributed data-intensive applications. *11th IEEE*

International Symposium on High Performance Distributed Computing, pages 352–358, 2002.

[192] Stefania Pandolfi. CERN data centre passes the 200-petabyte milestone. `https://home.cern/about/updates/2017/07/cern-data-centre-passes-200-petabyte-milestone`. Accessed: July 2017.

[193] Ariana Tantillo. Brookhaven lab's scientific data and computing center reaches 100 petabytes of recorded data. `https://www.bnl.gov/rhic/news2/news.asp?a=12061&t=today`. Accessed: June 2017.

[194] Costin Caramarcu, Christopher Hollowell, William Strecker-Kellogg, Antonio Wong, and Alexandr Zaytsev. The role of dedicated data computing centers in the age of cloud computing. *Journal of Physics: Conference Series*, 898(8), 2017.

[195] Ilka Antcheva, Maarten Ballintijn, Bertrand Bellenot, Marek Biskup, Rene Brun, Nenad Buncic, Ph Canal, Diego Casadei, Olivier Couet, Valery Fine, et al. ROOT — a C++ framework for petabyte data storage, statistical analysis and visualization. *Computer Physics Communications*, 180(12):2499–2512, 2011.

[196] Rene Brun, L Urban, Federico Carminati, Simone Giani, M Maire, A McPherson, F Bruyant, and G Patrick. GEANT: Detector description and simulation tool. Technical report, CERN, 1993.

[197] Disk pool manager (DPM). `https://svnweb.cern.ch/trac/lcgdm/wiki/Dpm`. Accessed: May 2015.

[198] Gaurav Khanna, Umit Catalyurek, Tahsin Kurc, Rajkumar Kettimuthu, P Sadayappan, and Joel Saltz. A dynamic scheduling approach for coordinated wide-area data transfers using GridFTP. In *Parallel and Distributed Processing (IPDPS)*. IEEE, 2008.

[199] Andrew Hanushevsky, Artem Trunov, and Les Cottrell. Peer-to-peer computing for secure high performance data copying. In *International Conference on Computing in High Energy and Nuclear Physics*, 2001.

[200] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming*. Elsevier, Amsterdam, 2006.

[201] Pavel Troubil and Hana Rudová. Integer linear programming models for media streams planning. *International Conference on Applied Operational Research*, 11:509–522, 2011.

[202] Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck, and Guido Tack. MiniZinc: Towards a standard CP modelling language. In *Principles and Practice of Constraint Programming*, pages 529–543. Springer, 2007.

[203] Guido Tack, Mikael Lagerkvist, and Christian Schulte. Gecode: an open constraint solving library. In *Workshop on Open-Source Software for Integer and Constraint Programming (OSSICP)*, 2008.

[204] Korea institute of science and technology information KISTI. `http://en.kisti.re.kr/`. Accessed: February 2015.

[205] Adrian Casajus, Ricardo Graciani, Stuart Paterson, Andrei Tsaregorodtsev, et al. DIRAC pilot framework and the DIRAC Workload Management System. *Journal of Physics: Conference Series*, 219(6), 2010.

[206] Iosif Legrand, Harvey Newman, Ramiro Voicu, Catalin Cirstoiu, Costin Grigoras, Ciprian Dobre, Adrian Muraru, Alexandru Costan, Mihaela Dediu, and Corina Stratan. MonALISA: An agent based, dynamic service system to monitor, control and optimize distributed systems. *Computer Physics Communications*, 180:2472–2498, 2009.

[207] Frederic Magoules, Thi-Mai-Huong Nguyen, and Lei Yu. *Grid Resource Management: Towards Virtual and Services Compliant Grid Computing*. CRC Press, Inc., 1st edition, 2008.

[208] Andrew V Goldberg. An efficient implementation of a scaling minimum-cost flow algorithm. *Journal of Algorithms*, 22:1 – 29, 1997.

[209] Rajkumar Buyya and Manzur Murshed. GridSim: A toolkit for the modeling and simulation of distributed resource management and

scheduling for grid computing. *The Journal of Concurrency and Computation: Practice and Experience (CCPE)*, 14, 2002.

[210] Barak Naveh et al. JGraphT. `http://jgrapht.sourceforge.net`. Accessed: September 2017.

[211] WLCG REsource, Balance & USage (REBUS). `https://rebus.cern.ch/`. Accessed: October 2015.

[212] LHC Optical Private Network (LHCOPN). `http://lhcopn.web.cern.ch/lhcopn/`. Accessed: December 2015.

[213] Anthony Sulistio, Gokul Poduval, Rajkumar Buyya, and Chen-Khong Tham. Constructing a grid simulation with differentiated network service using GridSim. In *6th International Conference on Internet Computing*, 2005.

[214] William Allcock, John Bresnahan, Rajkumar Kettimuthu, Michael Link, Catalin Dumitrescu, Ioan Raicu, and Ian Foster. The Globus striped GridFTP framework and server. In *ACM/IEEE conference on Supercomputing*. IEEE, 2005.

[215] Michal Zerola. *Distributed Data Management in Experiments at RHIC and LHC*. PhD thesis, Czech Technical University, 2012.

[216] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *ACM SIGCOMM Computer Communication Review*, volume 29, pages 251–262. ACM, 1999.

[217] Megiddo, Nimrod, and Modha. Outperforming LRU with an adaptive replacement cache algorithm. *Computer*, 37:58–65, 2004.

[218] Song Jiang and Xiaodong Zhang. Efficient distributed disk caching in data grid management. In *IEEE International Conference on Cluster Computing (CLUSTER'03)*, pages 446–451, 2003.

[219] Jagdish Prasad Achara, Abhishek Rathore, Vijay Kumar Gupta, and Arti Kashyap. An improvement in LVCT cache replacement policy for

data grid. In *13th International Workshop on Advanced Computing and Analysis Techniques in Physics Research*, 2010.

A List of publications

1. Dzmitry Makatun, Jérôme Lauret, and Hana Rudová. Planning of distributed data production for High Energy and Nuclear Physics. *Cluster Computing*, 2018. (Accepted).

2. Dzmitry Makatun, Jérôme Lauret, Hana Rudová, and Michal Šumbera. Provenance-aware optimization of workload for distributed data production. *Journal of Physics: Conference Series*, 898(5), 2017.
   http://iopscience.iop.org/article/10.1088/1742-6596/898/5/052038/meta

3. Dzmitry Makatun, Jérôme Lauret, Hana Rudová, and Michal Šumbera. Network flows for data distribution and computation. *Proceedings of the IEEE Symposium on Computational Intelligence in Scheduling and Network Design*, 2016.
   http://ieeexplore.ieee.org/abstract/document/7850083/

4. Dzmitry Makatun, Jérôme Lauret, Hana Rudová, and Michal Šumbera. Simulations and study of a new scheduling approach for distributed data production. *Journal of Physics: Conference Series*, 762(1), 2016.
   http://iopscience.iop.org/article/10.1088/1742-6596/762/1/012023/meta

5. Dzmitry Makatun, Jérôme Lauret, Hana Rudová, and Michal Šumbera. Model for planning of distributed data production. In *Proceedings of the 7th Multidisciplinary International Scheduling Conference (MISTA)*, pages 699–703, 2015.
   http://www.schedulingconference.org/previous/publications/displaypub.php?key=2015-699-703-A&filename=mista.bib

6. Dzmitry Makatun, Jérôme Lauret, Hana Rudová, and Michal Šumbera. Planning for distributed work flows: constraint-based coscheduling of computational jobs and data placement in distributed environments. *Journal of Physics: Conference Series*, 608(1), 2015.
   http://iopscience.iop.org/article/10.1088/1742-6596/608/1/012028A/meta

7. Dzmitry Makatun, Jérôme Lauret, and Michal Šumbera. Study of cache performance in distributed environment for data processing. *Journal of Physics: Conference Series*, 523(1), 2014.
http://iopscience.iop.org/article/10.1088/1742-6596/523/1/012016/meta