MISTA 2015

Distributed data production planning for High Energy and Nuclear Physics

Dzmitry Makatun · J´erˆome Lauret ·

Hana Rudov´a · Michal Sˇumbera

Abstract Modern experiments in High Energy and Nuclear Physics (HENP) heavily rely on distributed computations using multiple computational facilities across the world. One of the essential types of the computations is a distributed data production where petabytes of raw files from a single source has to be processed once using thousands of CPUs at distributed computational facilities and the output has to be transferred back to that source. The major factor limiting the performance of such computations is a latency caused by access to the remotely stored data. In this paper we propose a new scheduling approach for distributed data production which is based on the network flow maximization model having a polynomial complexity. In our approach a central planner defines how much input and output data should be transferred over each net- work link in order to maximize the computational throughput. Such plans are created periodically for a fixed planning time interval using the knowledge of the network struc- ture, its bandwidth and current status of computational sites. Our approach improves overall computational performance due to three factors: transferring input files in ad- vance before their processing, balancing of the network traffic and transferring files in a coordinated manner. The model was tested using one of the standard tools for Grid simulation (GridSim). Our simulations based on the log records from the data production framework of the experiment STAR have shown that the proposed model systematically provides a better performance compared to the simulated traditional techniques. Under certain conditions (such as limited bandwidth and/or existence of alternative transfer paths) the performance improvement can reach 28 %.

Dzmitry Makatun

Faculty of Nuclear Physics and Physical Engineering, Czech Technical University in Prague

E-mail: [makatun@rcf.rhic.bnl.go](mailto:makatun@rcf.rhic.bnl.gov)v

J´erome Lauret

STAR, Brookhaven National Laboratory (BNL), USA E-mail: [jlauret@bnl.go](mailto:jlauret@bnl.gov)v

Hana Rudova´

Faculty of Informatics, Masaryk University, Brno, Czech Republic

E-mail: [hanka@fi.m](mailto:hanka@fi.muni.cz)uni.cz

Michal Sˇumbera

Nuclear Physics Institute (NPI), Academy of Sciences (ASCR), Czech Republic

E-mail: [sumb](mailto:sumbera@ujf.cas.cz)era@ujf.cas.cz

1 Introduction

The STAR experiment [1] at the Relativistic Heavy Ion Collider (RHIC) studies a primordial form of matter that existed in the universe shortly after the Big Bang. Collisions of heavy ions occur millions of times per second inside the detector, producing tens of petabytes of raw data each year. All the raw data has to be processed in order to reconstruct physical events which are further analyzed by scientists. This process is called data production. Like any other modern experiment in High Energy and Nuclear Physics (HENP), STAR relies on distributed data processing, making use of several remote computational sites (for some experiments this number can scale up to many tens).

When running data intensive applications on distributed computational resources long I/O overheads may be observed as access to remotely stored data is performed. Latency and bandwidth can become the major limiting factors for the overall compu- tation performance and can reduce the CPU time / wall time ratio due to excessive I/O waiting. Widely used data management systems in the HENP community (Xrootd [26], DPM [23]) are focused on providing heterogeneous access to distributed storage and do not consider data pre-placement with respect to available CPUs, job duration or network performance. At the same time job scheduling systems (PBS [21], Condor [25]) do not reason about transfer overheads when accessing data at distributed storage. For this reason, an optimization of data transfers and distribution across multiple sites is often done manually, using a custom setup for each particular infrastructure [3].

In previous collaboration between BNL and NPI/ASCR, the problem of efficient data transfer in a Grid environment was addressed [22]. Data transfers between n computational sites and m data locations were considered but job scheduling was not covered by that work. In [15] we proposed a constraint programming planner that schedules computational jobs and data transfers in a distributed environment in order to optimize resource utilization and reduce the overall completion time. Since such global scheduling is computationally demanding it should be divided into several stages in order to improve scheduler performance and scalability. A planning of resource load can be completed before scheduling particular file transfers and jobs and this is our new proposal presented in this work. To summarize, we address the problem of data production planning, answering the question how the data should be transferred given the network structure, bandwidth, storage and CPU slots available. This will allow local schedulers to process jobs and have CPUs busy all the time while not exceeding disk and network capacities. We have presented the initial ideas of the model at the MISTA 2015 conference [14]. Since that time, the model has significantly evolved, it was implemented in code and tested using simulations with real data.

1.1 Related work

The general methods for scheduling parallel jobs with communication delays are ex- tensively described in [7] and [18]. A compact description of widely used job schedul- ing policies can be found in [12] and [19]. Those policies, such as First Come First Served (FCFS), conservative backfilling (CONS), aggressive backfilling (EASY), selec- tive backfilling, etc., are focused on maximizing CPU throughput and fairness to users. However, the above strategies are dedicated to manage a single computational cluster where data transfer overhead is not a significant factor. Also, for the data production

workflow such metrics as fairness to users is irrelevant, since all the jobs are typically submitted by the same user.

A wide perspective of modern concepts and challenges in distributed computing can be found in [10]. An overview of the algorithms for scheduling of data intensive application in a computational Grid is provided in [13]. Let us discuss the existing algorithms with respect to the considered use case.

The Adaptive Regression Method (AdRM) [8] is focused on predicting the perfor- mance of data transfer operations in network-bound distributed data intensive appli- cations 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 the work. The expected improvement is based on optimal selection of resources for scheduled tasks and their order. While this can be beneficial in the multiuser environment, where the files being processed have various possible sources and destinations, in the data production all the computational tasks are equivalent and available resources are typically heavily loaded most of the time. For this reason, swapping particular computational jobs between resources and changing their order cannot improve an overall performance.

Optimization of data intensive applications in Grid was studied in [16]. In this 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. However, this is not the case for data production, when each file has to be processed once. Similarly, the Storage Affinity [17] approach exploits data reutilization to improve the performance of the application in Grid. Again, as stated before, the data reuse is impossible for the data production problem.

In [6], the authors have modified existing heuristics to schedule parameter sweep applications with file I/O requirements (e.g. Monte Carlo simulations), and studied an impact of inaccurate performance prediction on scheduling. The authors consider job scheduling on heterogeneous resources (Grid) taking data transfer overhead for each job into account. While the input transfer overhead was estimated knowing an end-to- end connection speed, neither the file transfers were scheduled at network links, nor actual network topology was taken into account. In the case of the data production in HENP, uncoordinated data transfers may oversaturate the network capacity which leads to an overall degraded performance. For this reason in our research we consider a network load planning.

An explicit model distributing jobs over a Grid with respect to the network band- width was proposed in [4]. 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 time. In our study we do not limit the network topology to trees and assume fluctuations of job parame- ters. This factor has an increased value, since with the constantly increasing number of participating sites the hierarchical structure (implied by the tree network topology) of the Grid in HENP is not strongly held. Moreover, making the use of connections between computational nodes at the same network distance from the central storage can improve the overall throughput, as it will be shown in Section 6.5 of this paper.

To summarize, the existing scheduling policies for data intensive applications pro- vide performance optimization, which is based on either data replication, difference in job requirements to the resources or presence of multiple users with different connec- tivity to the Grid. However, those assumptions are inapplicable to the data production case. In addition to that, many of the approaches use a simplified network model, which is extremely crucial in the case when the network bandwidth becomes the main

bottleneck. For that reason, none of the existing algorithms meets all the specifics of the modern data production in HENP.

1.2 Summary of the proposed approach

In this paper we propose a model of distributed data production, where all the files from a single source has to be processed once and the results transferred back. Each data process- ing job requires exactly one CPU and each CPU executes a single job at a time. In our approach a central planner defines how much data of each type (input or output) 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 know- ing the structure of the network, its bandwidth and current status of computational sites (number of available CPUs, disk storage and the amount of input/output data). The logic of the planner is based on the network flow maximization which has a polyno- mial complexity, and due to that meets the need for scalability and reasonably fast plan generation. When the plan is created by the central planner, it is executed by handlers (dedicated services) acting at each computational site which participates in the data processing. Each handler is responsible for transferring data to other computational sites with respect to the plan and for the submission of jobs to CPUs.

As a summary of our contribution, we can state that, we have performed an analysis of distributed data production in HENP and proposed a new model which considers both job scheduling and data transfers. In contrast to the known approaches, our model takes an advantage of the specifics of the distributed data production and in- cludes a realistic network model. To achieve efficiency and adaptability, the overall plan is computed cyclically and is distributively processed by the local scheduler at each computational node. Solution for each cycle is based on the network flow maximization approach which has a polynomial complexity. We have implemented the data produc- tion planner based on our model and a simulating environment using a standard Grid simulation framework (GridSim). We have verified the model with simulations based on data obtained from the real data production framework of a HENP experiment, which have shown a significant improvement compared to currently used scheduling techniques.

The paper is organized as follows: in Section 2 we provide a detailed description of the problem, Section 3 presents our mathematical model, then the algorithms for a plan generation are given in Section 4, Section 5 describes how the plan execution is implemented, and, finally, Section 6 provides a description of the simulation setup, used data, simulated scheduling approaches and the obtained results.

2 Problem description

We consider data processing in a computational Grid, which is a collection of computer resources from multiple locations. We focus on a specific type of data processing which is typical for computations in HENP and is called a data production. In general, our goal is to maximize a computational throughput of such data processing. In this section we will describe specifics and important elements of the

considered problem, which are a central storage, computational nodes, interconnecting network and computational jobs.

The central storage is the data storage where all the data of the physical experiment is being persistently stored. It is usually located at the so called Tier-0 site, which is the main computational facility of the experiment. The raw data from the detector is being automatically archived to the central storage in the form of separate files (typi- cally of several gigabytes of size) and after that it 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 (usually several gigabytes of size) which we refer as output files. All the reconstructed data are being persistently stored at the central storage as well. In most of the experiments the central storage is imple- mented as a tape storage coupled with a faster disk one, however its internal mechanics is beyond the scope of the model. In order to address possible bottlenecks related to the central storage we consider two values: k0 is the total size of currently available input files and k0 is the currently available free space to store new reconstructed data (output files).

In our problem a (computational) node is one or a set of closely connected machines (also can be referred as a computer cluster, or site) which has a fast access to a common data storage (referred as a local disk ) and a shared connection to the outer network. A computational facility of a scientific institution is an example. Often, an institution provides access to only a fraction of its computational resources (i.e. a fixed number of CPUs and limited storage size) to be used by an experiment. In such case, a set of those granted resources can be considered as a computational node. The key principle to identify a set of machines as a computational node 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 Grid. We assume that there is a common job scheduling system (local scheduler) which allows to submit jobs to all of the CPUs of the machines combined to a node. We also assume that all of the CPUs at a particular node have the same processing characteristics, but they can differ between nodes. Each CPU is considered as a separate processing element, which can correspond to a single core in modern architectures. The CPUs are modeled as a space-shared resource, which means that each CPU can execute a single job at a time. In our model each computational node i is described by the number of CPUs N C P Ui , average processing speed of its single CPU αi and available local disk space Diski . Also, there is a local queue of the input files which are already transferred to this node, but not submitted for the processing yet. The input files from the local queue can be submitted for processing or transferred

to another node if needed. The total size of those files is I in . Similarly, the total size

i

of output files of previously finished jobs which are waiting to be transferred out of the node is I out . The total size of the input files that are currently being processed is denoted C in and the total disc space which is reserved for output of currently running jobs is C out . The data on the node status can be obtained from its monitoring system, in addition to the above parameters it also includes the number of CPUs occupied by running jobs N C P U busy and free space at the local disc Fi .

i

i

i

i

The computational nodes are interconnected to each other and to the central stor- age with network links of a given bandwidth b(l) (l is an id of a link) so that they form a Grid, which can be described a by a weighted graph. Obviously, the graph must be connected. Such graph can represent a realistic network topology, where the network routers can be represented by computational nodes with zero number of CPUs and local disc size. In our model we consider only the network propagation latency, which

can be calculated as a size of transferred data divided by the bandwidth of the link. In realistic case, the network links are full-duplex, which implies the same bandwidth in both directions, however, this is not a strong requirement for our solution.

The data production in HENP has a data level of parallelism, which means that it can be (and actually is) divided into independent computational jobs applying the same processing on different subsets of data. In other words, each computational job has its unique input file, uses a single CPU and produces a unique output file. There is a one to one relation between jobs, input and output files in data production. In our problem the size of input file of a job j is denoted as I nSizej , the size of output file is OutSizej and the job duration is pj . Only the size of the input file is known in advance, before the job is finished. However, the two other parameters are related to it by the following expressions: pj ≈ αi · I nSizej and OutSizej ≈ β · I nSizej , where i is an id of a processing node, αi and β are considered as constant coefficients. Since all the jobs perform the similar type of processing, knowing an average values of αi and β for previously completed jobs we can estimate the size of output and a duration of a particular job. From the scheduling system view, all the data production jobs are submitted by a single production user, for this reason such metric as ”fairness to users” is not related to the considered problem. Also, there is no any requirement on the order of particular jobs.

As mentioned before, the main goal is the maximization of an overall computational performance. In a real environment the performance can be estimated by such param- eters as a total CPU utilization (given that there is no job duplication and I/O waiting CPU cycles are not counted as useful work) or makespan of processing of a given set of input files (a dataset), i.e. the completion time of the last output file transfer.

3 Model based on network flow maximization

Let us consider a scheduling time interval ∆T . We assume that at the starting moment some of the CPUs in the Grid are busy, 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 time interval ∆T in order to avoid draining of the local queues by the end of this interval.

The computational Grid is represented by a directed weighted graph where vertexes ci ∈ C are computational nodes and edges l ∈ L are network links. The weight of each link b(l) is the amount of data that can be transferred over the link per unit of time (i.e., bandwidth). One of the nodes c0 is the central storage where all the input files for the further processing are initially placed. All the output files has to be transferred back to c0 from the computational nodes. We will give two separate problem formulations: for an input and output transfer planning based on network flows.

In order to formulate a network flow maximization problem [2] for input/output file transferring we have to 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 ∈ E with their capacities cap(e), representing the amount of data which can be transferred during ∆T . A solution that assigns a non-negative integer number f (e) to each edge e ∈ E can be found in polynomial time with known algorithms. Here f (e) is understood as an actual amount of data to be transferred within ∆T .

3.1 Input flow planning

In order to transform a given graph of a Grid into a capacitated {s, t} network for an input transfer problem (see Figure 1) we add two dummy vertexes: a source s and a sink t. Next we add dummy edges di ∈ D from each computational node i to the sink, and a dummy edge q0 from the source s to the central storage c0 . These dummy edges allow us to introduce constraints on the storage capacity and CPU throughput of the computational nodes. The set of vertexes V consists of computational nodes C and dummy vertexes: V = C ∪ {s, t}. The final set of edges consists of real network links L, dummy edges D from computational nodes to the sink and from the source to the central storage q0 : E = L ∪ D ∪ {q0 }. Capacity of each edge defines the maximal amount of data that can be transferred over an edge within time interval ∆T :

 b(e) · ∆T if e ∈ L

cap(e) =  wi if e = di ∈ D

 k0 if e = q0

(1)

where wi is the maximal amount of new data that can be stored at the node i for processing without exceeding its storage capacity or CPU throughput, k0 is the total size of input files currently available to be transferred from c0 . We denote the solution for the input transfer problem as f in (e). It specifies the amount of input data that should be transferred over each link during ∆T .

3.2 Output flow planning

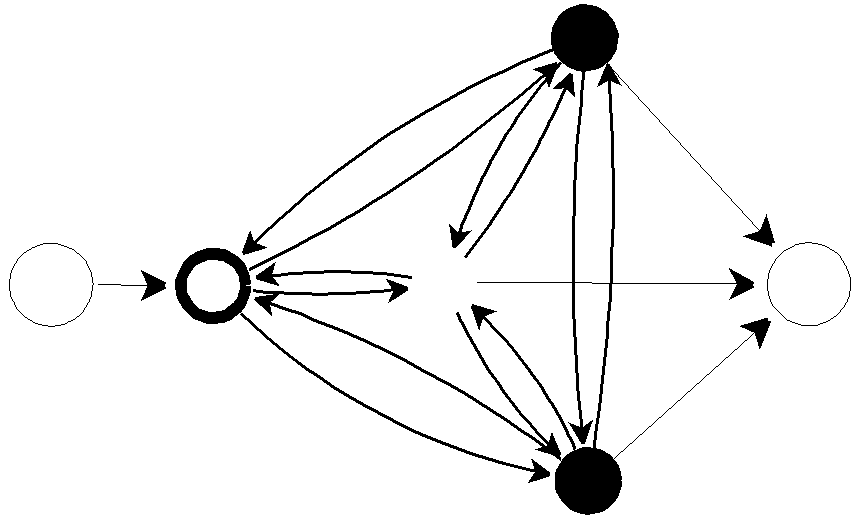
For transfer of output files we use a similar transformation, but swap the source s and the sink t, change the direction of dummy edges and redefine capacities of dummy edges (see Figure 2). In this case the capacity k0 of the dummy edge q0 leading from the central storage c0 to the sink s is equal to the amount of data which can be transferred to c0 within time interval ∆T (it is limited by the available space at the central storage). The capacity wi of dummy edges di leading from the source t to computational nodes ci is equal to the maximum amount of output data which can be transferred from the

c1

source c0

q0

. . .



sink

*s t*

ci

Fig. 1 A data production Grid represented as a capacitated {s, t} network for the input transfer planning (general case). c0 is the central storage, ci are computational nodes (where i > 0), solid lines are network links L, dotted lines are dummy edges, q0 is a dummy edge leading from the source s to c0 .

node ci .

 b(e) · ∆T if e ∈ L

cap(e) =  wi if e = di ∈ D

 k0 if e = q0

αi

(2)

We denote the solution for the output transfer problem as f out (e). It specifies the amount of output data that has to be transferred over each link during ∆T .

3.3 Capacities of dummy edges

Let us consider data production jobs which perform the same type of data analysis and have similar characteristics. In this case we can assume that the duration pj of a job j processing an input file of size I nSizej at a node i is pj = αi · I nSizej where αi is constant for each node i. Also, the ratio of size of input I nSizej and output OutSizej files of each job j is considered to be constant for the same type of data processing, i.e., OutSizej = β · I nSizej . During the time interval ∆T a node i with N C P Ui of

CPUs will process 1

α

i

· N C P Ui · ∆T of input data and will produce β

· N C P Ui · ∆T

of output data.

Capacities of dummy edges define the maximum net amount of input/output data that can be transferred to/from the node respectively. Let us consider transfer of output files first. Let I out be the initial size of output data at a local storage which are ready to be transferred (output data of previously finished jobs). We also need to estimate the amount of new output data that will be created during the planning time interval

i

∆T (output data of jobs that will finish during ∆T ). If most of the CPUs at the node

are saturated, then this value can be estimated as β

α

i

· N C P Ui · ∆T . In the opposite

case, if not all the CPUs are busy, such estimation may become exaggerated. However,

the considered value cannot be greater than the total size of output files of currently running jobs C out , because ∆T is assumed to be significantly smaller than an average job duration, i.e. it is unlikely that a job started within ∆T will also be finished within it. To find the bottleneck value, we have to select the smallest estimation, and therefore, the final formula of the capacity of dummy edges for output problem is:

i

wi = I out β

out

i + min(

α

i

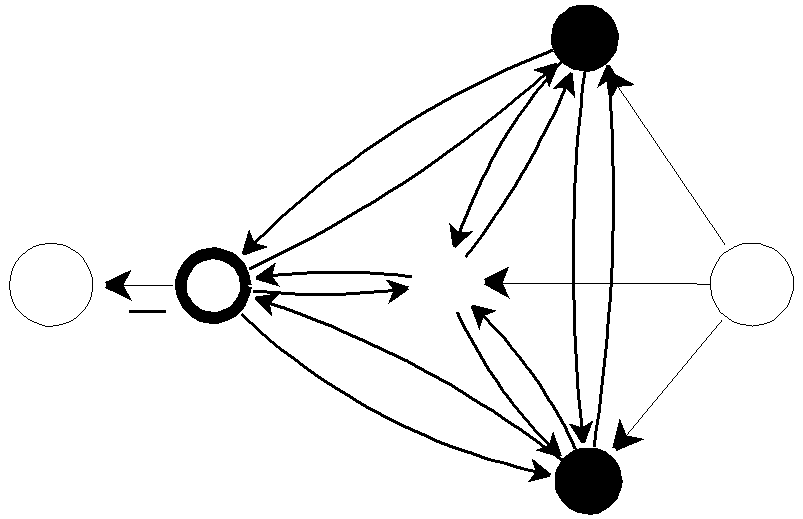
· N C P Ui · ∆T , Ci ) (3)

c1

sink c0

*s* q0

. . .



source

*t*

ci

Fig. 2 A data production Grid represented as a capacitated {t, s} network for the output transfer planning (general case). c0 is the central storage, ci are computational nodes (where i > 0), solid lines are network links L, dotted lines are dummy edges, q0 is a dummy edge leading from c0 to the sink s.

Next we will consider transfers of input files. Let us denote the available free disk space at the node i as Fi . In order to avoid running out of free space at the local disk, we need to set an upper limit δ for a planned disk usage. That ensures that during the plan execution there is always enough space for an output file of a new job or for a new incoming file. This is especially important, since job duration and file sizes can fluctuate from the average values which are used for the plan creation. In other words, we assume that there is a buffer of size Diski · (1 − δ) which should be planned to remain free.

The amount of input data which can be transferred to the processing node is limited by two factors: size of the local disk and the CPU throughput. Obviously, we should not transfer to a node more data than it can store and process, but at the same time we should aim to reach the peak throughput of the system. For this reason, it is important to identify, which factor is a bottleneck in any given case. We have to consider two possible cases: when all the CPUs at the node are saturated (N C P U busy = N C P Ui )

i

and when there are free CPU slots (N C P U busy < N C P Ui ).

i

In the first case the maximal amount of data that we can store to the disk is its free space minus buffer size (Fi − (1 − δ) · Diski ). At the same time, in order to adjust to CPU throughput, the total size of preplaced input files I in should not become greater than the total size of input files which are currently being processed (C in ), because

i

i

it is unlikely that all the jobs will be finished within ∆T . For this reason we should transfer additional C in − I in at maximum.

i i

In the second case, it is important to notice that if there are free CPUs at the

node, each incoming input file will be immediately submitted for processing, creating a reservation for a new output file of size OutSizej = β · I nSizej . For this reason, the constraint on the storage size gives us Fi −(1−δ)·Diski . From the other hand, in order to saturate unused CPUs, we need to transfer at least one file for each of them. If an

1+β

average size of an input file is fbin , then the required amount of input data would be

N C P Ui · fbin − I in .

i

In both cases, ether N C P U busy = N C P Ui or N C P U busy < N C P Ui we should

i i

select the minimal value between the storage and CPU constrains, in order to find the

bottleneck. The above discussion leads us to the final expression for the capacity of the dummy edges for the input problem:

( min(Fi − (1 − δ) · Diski , C in − I in ) if N C P U busy = N C P Ui

i i

wi =

min( Fi −(1−δ)·Diski in

i busy

(4)

1+β , N C P Ui · fbin − Ii ) if N C P Ui < N C P Ui

In the Eqn. 3 and 4 ∆T and δ are parameters of the scheduler. The other values used in Eqns. 3–4 can be obtained from monitoring data right before each planning iteration.

4 Solving Procedure

It can be proven that the maximum flow problems for input and output transfers can be solved independently under assumptions: (a) all the real network links in the considered Grid are full-duplex, i.e., a network throughput between two nodes is the same in both directions (b) in a steady state the size of the output transferred from

each node is proportional to the size of the input transferred to that node in each scheduling interval, i.e., f out (di ) = β · f in (di ), where β ≤ 1.

Since in real environment the assumption (b) will not strongly hold due to resource performance fluctuations we propose the following approach to solve the problem and create a single plan for a period ∆T :

1. Calculate values for wi using Eqn. 3 and current monitoring data.

2. Solve the problem for output data flows to obtain f out (e).

3. For real links l ∈ L reduce the capacity by the amount which is used by output transfers: cap(l) = b(l) · ∆T − f out (l).

4. Calculate values for wi using Eqn. 4 and current monitoring data.

5. Solve the problem for input transfers with wi and cap(l) defined in previous steps.

Find input data flows f in (e).

This procedure called scheduling cycle has to be repeated every time interval ∆T until all the data is processed and the output is transferred to the central storage. The main reason why we prioritize output files is that by transferring them we also create a free space for upcoming input files. 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.

To summarize, this procedure is expected to compute feasible data transfers such that as many CPUs in Grid as possible are busy with computational jobs while not exceeding local disk capacities.

5 Plan execution and job scheduling

When the plan for each period ∆T is created it has to be executed by the computational nodes. We assume that there is a dedicated service running at each node which is responsible for sending statistics to the planner, receiving the plan and executing it. We call such a service a local handler.

The local handler keeps counters of how much data of each type (input and output) remains to be send from this node to the neighboring nodes during the current plan iteration. This implies two counters (input and output) for each link connected to the node. When the handler receives a new plan it updates the counters to be equal to the flows of corresponding edges. In other words, a handler is responsible for transfers over outgoing edges of its node. If the handler has not managed to send/receive all the data according to the plan within the time window, the system will automatically recover from such an error, since each scheduling cycle rely on the current state of the system, but not on previously issued plans.

For the plan execution, each time when a new file arrives to the node, the handler decides either to keep the file for local processing or to forward it to another node. In order to make the decision, the handler executes the first of the options which requirements are satisfied:

1. If the received file is of input type and if there is a free CPU at this node then the file is submitted for processing.

2. 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. The counter is decreased by the size of the file.

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

Such an order of options ensures that the file will be processed as soon as it arrives to the node with a free CPU, and all the CPUs will be busy as long as there are unprocessed input files at the node. Since an input file will not be forwarded unless all the CPUs are busy, no excessive transfers will happen in the system.

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

The job scheduling logic of a handler can be described as simple as ”whenever there is a free CPU at the node select the next input file from the local disk (if any) and submit it to that free CPU”. The input files at the local disk are submitted for processing in the FIFO (First In First Out) order, since for a centrally submitted data production the order of particular jobs is irrelevant.

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

6 Simulations and its results

In our experiments the ”Grid Simulation Toolkit for Resource Modelling and Applica- tion Scheduling for Parallel and Distributed Computing” (GridSim) [5] was used for realistic simulations of distributed data production. It is a discrete event simulation toolkit which provides extensive models of computational clusters, job scheduling and execution, network, data transferring, etc. It was developed by the Cloud Computing and Distributed Systems (CLOUDS) Laboratory of the University of Melbourne. We built our new simulator based on experiences we have with GridSim which helped us to create efficient simulations [11]. In this work we have implemented an additional func- tionality for the plan execution, statistics collection and a simple storage management on top of the GridSim package for our simulations.

The planner was implemented in Java using a JGraphT [27] library which pro- vides EdmondsKarp algorithm [2] for solving the network flow maximization problem. The simulations were running under Windows 8 64-bit with java 1.8.0 60 (64b) on a computer with Intel i5 (4 cores) 2.50 GHz processor and 6 GB of memory installed.

6.1 Simulation setup

Our data comes from the remote data production of the experiment STAR which was ongoing at the computational facility of the Korea Institute of Science and Technology Information (KISTI) [24] for seven months in 2014. The log database of the data production framework contains information on job durations, sizes of input and output files. Records of 60,000 computational jobs were taken from the log and used as input for our simulations. The average parameters of this set of jobs are provided in Table 1.

Table 1 Parameters of 60,000 data production jobs recovered from the log database of the data production framework and used for simulations. The data production was setup at the KISTI computing facility for the STAR experiment and continued for 7 months in 2014.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Parameter | Units | Average | Min | Max | Total |
| Duration | hours | 46 | 12 | 64 | 314 (years) |
| Input file size | MB | 4320 | 895 | 4780 | 259 (TB) |
| Output file size | MB | 3022 | 105 | 3614 | 181(TB) |
| α | s/MB | 38 | 9 | 49.2 | - |
| β | - | 0.7 | 0.3 | 0.79 | - |

In order to setup realistic parameters of simulated computational nodes, two sources were considered: the data production framework at KISTI [9]; and the online monitor- ing system of the ALICE experiment at CERN (MonAlisa) [28]. According to these sources, the number of CPUs available at the remote nodes can vary from several units up to several thousands, while typical values for interconnecting network bandwidth are approximately 50 Mbps to 2 Gbps. More detailed setup descriptions are given separately for each particular simulation.

The planning time interval ∆T should be short enough in order to provide better adaptability to the current state of the nodes, at the same time, it should be long enough so that the planned data flows meet the discrete nature of file transfers. After a set of testing simulation runs it was set to 6 hours, and the upper limit for the disk usage δ was set to 95 %. All the CPUs are considered to be of the same processing speed. Parameters αi (equal for all nodes i, because they have the same processing

speed), β and fbin (the average size of input files) are set to the average values from

the Table 1. In all of the simulations, the input data is initially placed at the central storage, all the CPUs and local disks in the system are free at the initial moment of time.

To give a better idea behind our next considerations about the simulations, we will summarize the rest of this section. In Section 6.2 we will discuss the used network models, in Section 6.3 we will introduce simulated scheduling approaches, in Section

6.4 we will discuss testing simulations with a single remote node mimicking one of the real data production frameworks, and, finally, in Section 6.5 we will provide results of simulations with multiple remote nodes testing all the aspects of the planner in a realistic environment.

6.2 Data transfer modes

Sending files over the network is an essential part of our simulations in GridSim. It is briefly described below, more details can be found in [20]. When a simulated entity (i.e. computational node) executes a ”transfer the file” command, the file is stored to the output queue and then it is processed by the underlying network model. If there are several network links connected to the entity, then each of them has its own queue. We have used two file transfer modes 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 an 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.

An important consequence of the difference between the two modes appears when an entity sends a set of multiple files to the same destination simultaneously. In both modes the complete transfer time of the file set will be the same, but in the sequential transfer mode the files will arrive one by one and the first file will become available faster than in the parallel transfer mode, where the delay of the first file will depend on size of the set. A comparisonal study of parallel and sequential transfer modes in real network can be found in [22], where the authors has shown that transferring files sequentially (but using multiple threads) can be advantageous for HENP computations compared to parallel transfer of multiple files. However, the parallel transfer mode is a more common model for 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.

6.3 Simulated scheduling approaches and their comparison

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 this paper. The se- quential transfer mode is used as the preferable one.

– PUSH par: Whenever there is a free CPU in the Grid, the next input file is sent from the central storage to the computational node with this free CPU. When a node 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 is informed that a CPU became free at the sender node. In such a manner, input files are being sent from the central storage until all the data are processed. The shortest network path is used for file transfers. The parallel transfer mode is used here, which models uncoordinated concurrent access to remote files by simultaneous jobs. This scheduling approach corresponds to the distributed data production setup in many HENP experiments, in particular, data production at KISTI for the STAR experiment [9].

– PUSH seq: The job scheduling in 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.

– no network: Again, the job scheduling is the same as in two previous approaches, but the network is not considered here. The production is performed as if all the CPUs are aggregated at the single site and can read/write the data directly from/to the storage with no latency. These simulations serve as a base case for the compar- ison. It allows to estimate the limit for the processing makespan, and the influence of the transfer latency. Obviously, none of the other scheduling approaches can process the data faster than the one with no network delay.

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 from the central

storage until the finish time of the last output file transfer to it. For convenience, in the following sections, all the presented makespan values are divided by a fixed value which is the makespan of the no network approach of a selected case (in other words, the makespan value of no network approach is selected as a unit makespan). Selection of such unit makespan value is explicitly explained in each case. This allows to understand better how the network latency influences the computational performance.

For a comparison of two scheduling policies, a makespan improvement can be cal- culated as follows:

makespan improvement = makespan1 − makespan2

makespan1

(5)

6.4 Simulations with a single remote node

In the first set of simulations we aimed to reproduce the remote data production of the experiment STAR at the KISTI computing facility. The environment consisted of a single remote computational node connected to the central storage. While in the real setup the number of CPUs at the remote site was 1,000 CPUs and the network bandwidth was approximately 2 Gbps [9], we did simulations with a wide scope of these parameters in order to study their influence on the computational efficiency. By simulations with a single remote node we would like to address the following questions. Is the proposed model able to provide an efficient data pre-staging at the remote node? How does the required bandwidth depends on the number of CPUs at the remote site? How many CPUs at the remote site can be exploited efficiently, given the network bandwidth? It also allows us to better study the problem and confirm the validity of all the simulated scheduling approaches.

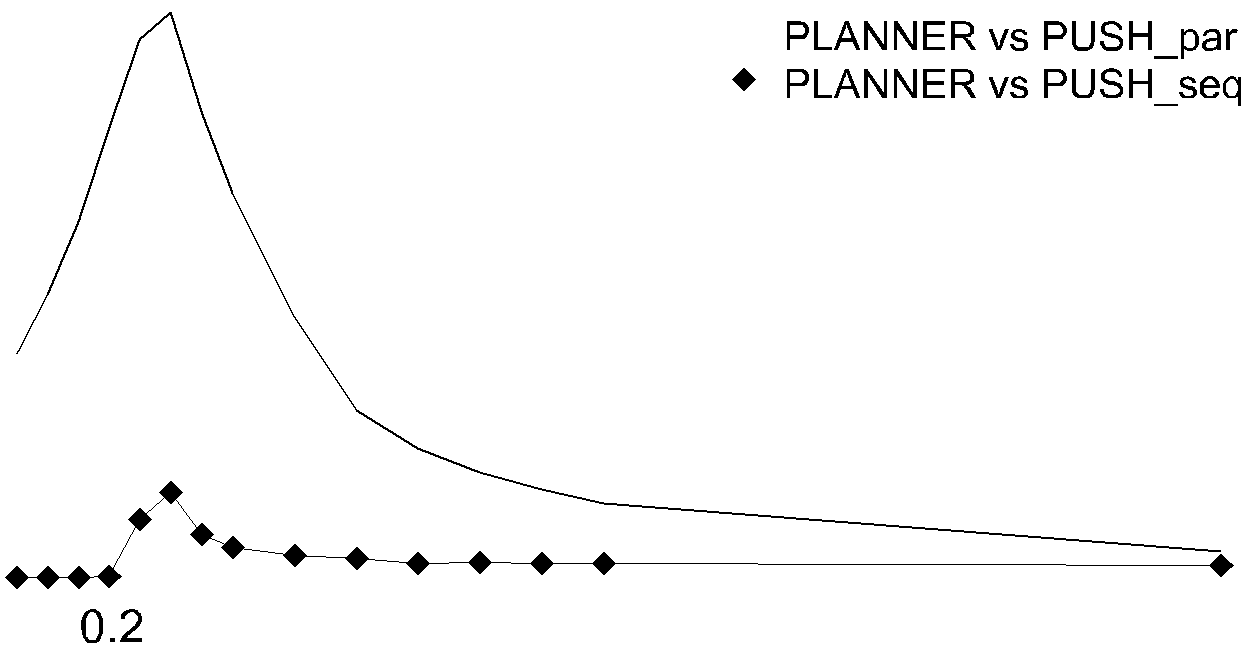
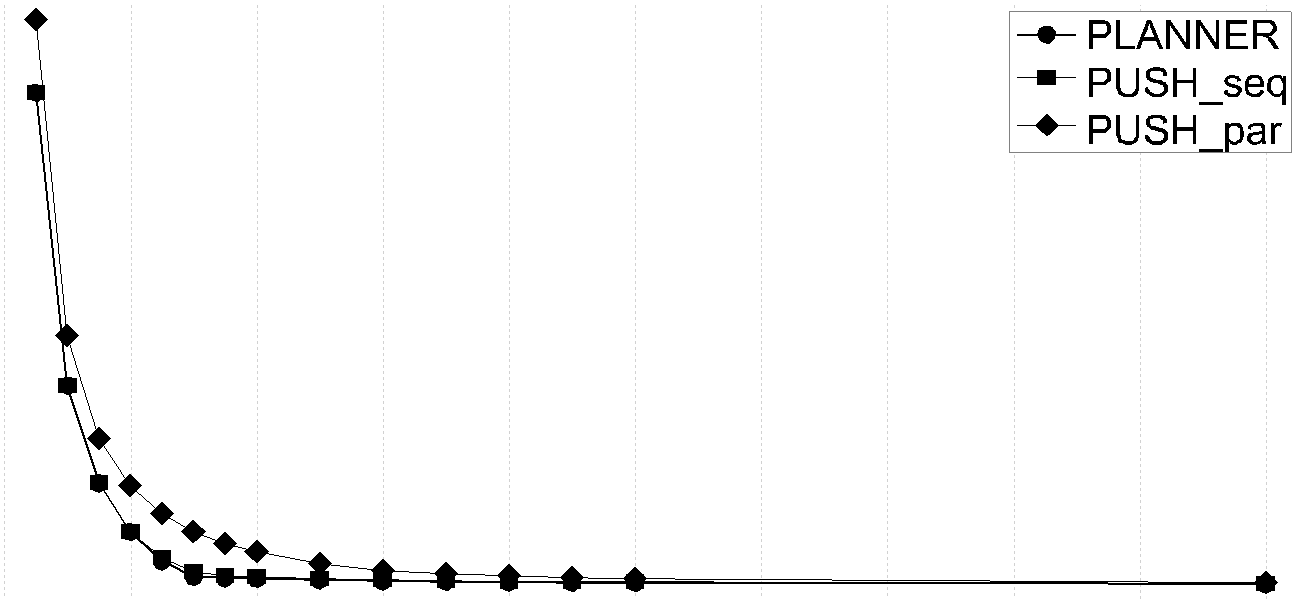
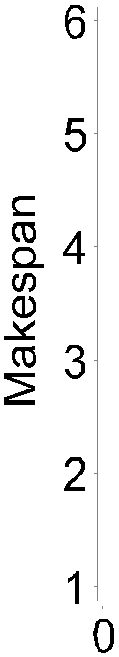
The Figures 3 and 4 present results of simulations where the number of CPUs at the remote site was 1,000, the size of its storage was 12 TB and the network bandwidth was changing from 50 Mbps to 2 Gbps in different simulation runs. A subset of the last

7,000 job records from the data production framework was used (see parameters of the entire dataset in Table 1). The total size of input files was approximately 25 TB. This amount of data is sufficient for the given number of CPUs to demonstrate the behavior of the simulated scheduling algorithms. The plot at Figure 3 shows the dependence of the makespan on the network bandwidth for three scheduling approaches (PLANNER, PUSH par and PUSH seq, see Section 6.3). For convenience, the makespan is given in units equal to 9 days, 15 hours, 39 minutes and 13 seconds which is the makespan of the no network approach (obviously, it does not depend on the bandwidth). As it can be observed from the plot, when the network bandwidth is 2 Gbps or greater, the I/O transfer latency for 1,000 CPUs becomes negligible compared to the job duration, for that reason all the scheduling models deliver the makespan close to the no network case. In the opposite case, when the network bandwidth is smaller than 300 Mbps, it is not sufficient to saturate available CPUs with data, as the result the makespan increases dramatically for all the scheduling policies. To better distinguish amongst particular scheduling approaches, the makespan improvement of the proposed model against others is given at Figure 4. The proposed model can provide a significant improvement (reaching

28 %) compared to the scheduling approach used in practice (PUSH par). Also in the case, when the network bandwidth is not sufficient to keep all the CPUs busy, the PLANNER scheduling approach permanently provides a better performance. Using this plot one can conclude that in the case of a single remote site the main improvement is



Fig. 3 Dependence of the data production makespan on the network bandwidth using a single remote node. The makespan is given in units equal to 9 days, 15 hours, 39 minutes and 13 seconds which is the makespan of the no network approach.



|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  | |  | | | | |
|  | |
|  |  |  |  |  |  |
|  | |
|  | |  | |  |  |  |  |
|  |  |  |  |  |  | |  | |  |  |  |  |
|  |  |  |  |  |  | |  | |  |  |  |  |
|  |  |  |  |  |  | |  | |  |  |  |  |
|  |  |  |  |  |  | |  | |  |  |  |  |
|  |  |  |  |  |  | |  | |  |  |  |  |

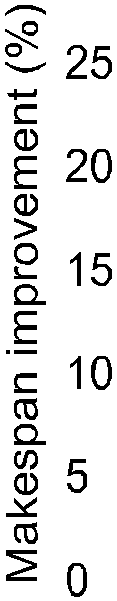


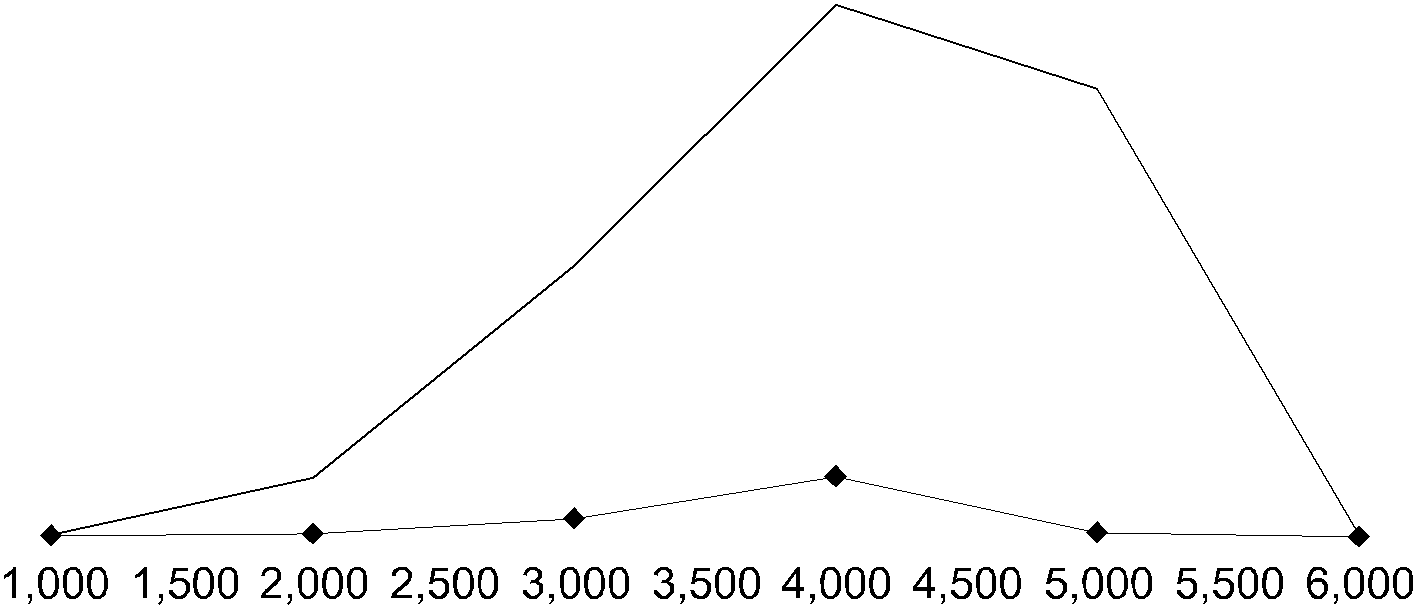
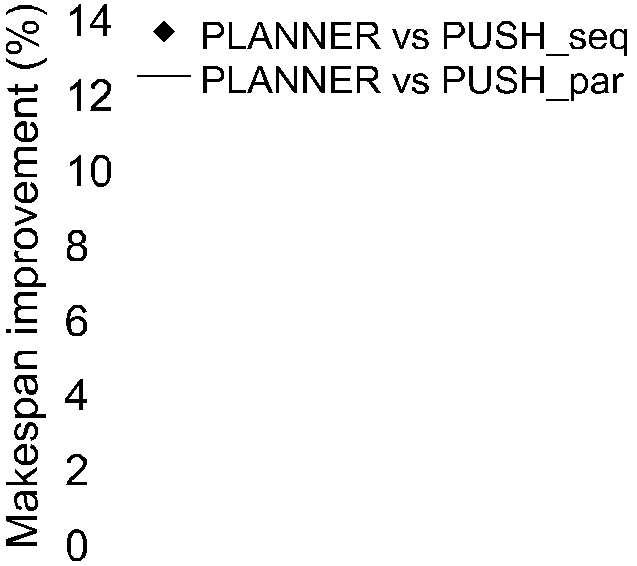
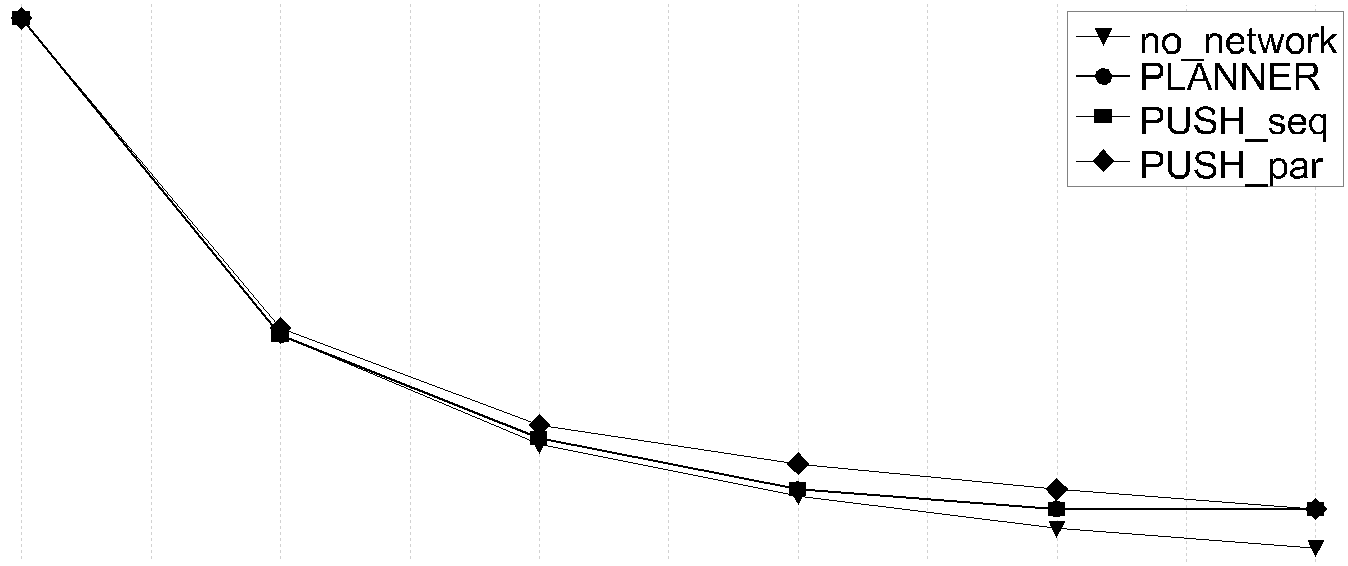
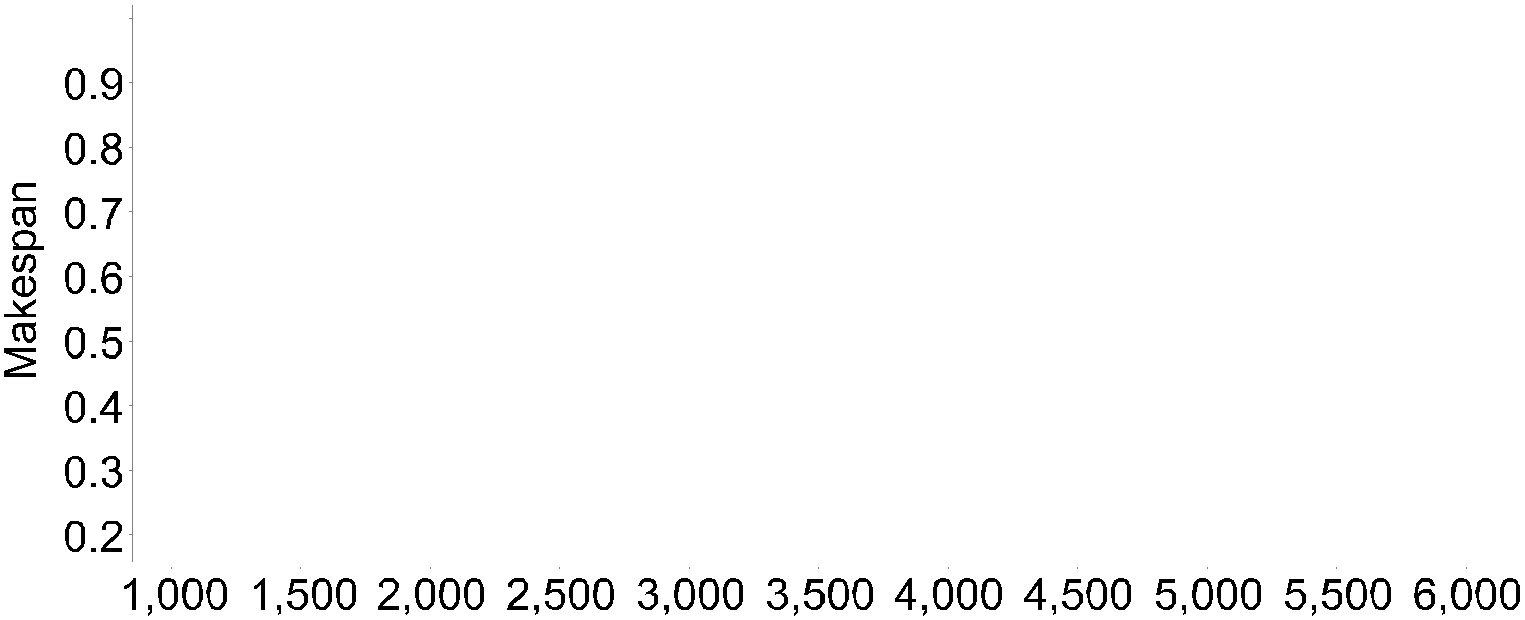
Fig. 4 A makespan improvement of the proposed model compared to other simulated schedul- ing approaches.

achieved due to sequential file transferring, while transferring file in advance encounters for up to 4 % of the makespan improvement.

Another set of simulations was performed with a fixed network bandwidth (1 Gbps) and a changing number of CPUs at the remote site (1,000 – 6,000). The size of storage was adjusted to the number of CPUs (10 TB for every 1,000 CPUs). The entire set of 60,000 computational jobs was used for these simulations. The plot at Figure 5 shows the dependence of the makespan on the number of CPUs at the remote site for all the scheduling approaches. This time the makespan unit is set to the makespan of the no network approach with 1,000 CPUs which is 116 days, 4 hours, 16 minutes and 2 seconds. With the growing number of CPUs the makespan decreases inversely linearly for the no network case, while for the other strategies the limited network bandwidth decreases the gain from using more CPUs. For 1,000 CPUs the network bandwidth of 1 Gbps does not produce any significant latency and all the scheduling algorithms produce the same result. At the right side of the plot, when the number of CPUs is 6,000, the performance is limited by the network bandwidth and additional CPUs do not decrease the makespan, and, consequently, none of the scheduling policies



Fig. 5 A dependence of a data production makespan on the number of CPUs available at the remote node which is connected over 1 Gbps link to the central storage. The makespan unit is set to 116 days, 4 hours, 16 minutes and 2 seconds which is the makespan of no network approach with 1,000 CPUs.



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  | | | |  |  |  |  |  |  |  |  |
|  | |  |  |  |  |  |  |  |  |
|  |  | |  |  |  | |  |  |  |  |  |  |  |
|  |  | |  |  |  | |  |  |  |  |  |  |  |
|  |  | |  |  |  | |  |  |  |  |  |  |  |
|  |  | |  |  |  | |  |  |  |  |  |  |  |
|  |  | |  |  |  | |  |  |  |  |  |  |  |
|  |  | |  |  |  | |  |  |  |  |  |  |  |
|  |  | |  |  |  | |  |  |  |  |  |  |  |



Fig. 6 A makespan improvement of the proposed model compared to other simulated schedul- ing approaches as a function of CPUs at the remote site.

can provide any improvement. The difference between the approaches is better visible at the plot at Figure 6 which depicts the makespan improvement of the PLANNER compared to two other models. For a number of CPUs between 1,000 and 6,000 the PLANNER approach achieves a better makespan than the one currently used in data production (PUSH par), allowing to use more CPUs efficiently. It achieves up to 14 % of the makespan improvement. The comparison against PUSH seq shows that the main part of improvement is achieved by the sequential file transferring, while transferring files ahead of processing (provided by the PLANNER approach) is responsible for up to 1.8 % of additional makespan improvement (which is approximately 50 hours in the simulated case).

At this point the reader might be confused by the very close behavior of the PLAN- NER and PUSH seq models, however, this is due to the simulation with a single remote site, while the planner was designed to gain an improvement not by only transferring files in advance and in a sequential mode, but also due to distribution of a network load in a more complex Grid topology. Such a case will be studied in the following subsection. However, the simulations with a single remote node are important to test the planner, and understand the dynamics of the process. These simulations allow us

FAST

1 Gbps

0.1 Gbps

CENTRAL STORAGE

0.1 Gbps

0.3 Gbps 0.1 Gbps

MEDIUM

0.1 Gbps

SLOW

Fig. 7 The simulated Grid with multiple computational nodes. Each node has 1,000 of CPUs and a local disc of 12 TB size. All the nodes have direct connections to the central storage (solid lines). In addition to that, all the nodes are interconnected with links of equal bandwidth (the doted lines).

to conclude, that the proposed model can utilize more remote computational power using a given network bandwidth and also can increase the performance in the case when the network bandwidth becomes decreased due to other network activity. These two factors provide more flexibility when setting up a remote data production.

6.5 Simulations with multiple remote nodes

The next set of simulations is dedicated to test the planner with a Grid of reasonable size. The Grid network for these simulations is depicted at Figure 7. It consists of three computational nodes named FAST, MEDIUM and SLOW which are connected to the central storage with a 1 Gbps, 300 Mbps and 100 Mbps network links, respectively. Each node has 1,000 of CPUs and a local disc of 12 TB size. In addition to that, all the nodes are interconnected with links of equal capacity (see perimeter links at Figure 7). The capacity of those links is changing equally from 10 to 500 Mbps in different simulations in order to study how the usage of indirect transfer paths can improve the data production efficiency. The PUSH par and PUSH seq models (see Section 6.3) are able to use the direct connections only, while the PLANNER considers all the available transfer paths. The set of 60,000 computational jobs was used for these simulations. Despite the simulated Grid does not mimic a particular real system, its parameters were chosen with respect to monitoring data of the computational Grid of the ALICE experiment [28]. Such a setup allows to test how our scheduling approach will perform in a realistic distributed environment.

The results of the simulations are given at Figure 8. The plot shows how the makespan improvement of the PLANNER compared to PUSH par and PUSH seq mod- els is depending on the bandwidth of the links between the remote nodes. As it is seen at the plot, the planner has reached up to 28 % improvement against the current schedul- ing approach (PUSH par) and up to approximately 20 % improvement against the PUSH seq. Additional monitoring has shown that the planner, as expected, redirects a part of the input flow from the ”FAST” node to the ”MEDIUM” and ”SLOW”; and from ”MEDIUM” to ”SLOW”. This allows to balance the network load and achieve

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | |  | |  |  |  |  |
|  |  |  |  |  |  | |  | |  |  |  |  |
|  |  |  |  |  |  | |  | |  |  |  |  |
|  |  |  |  |  |  | |  | |  |  |  |  |
|  |  |  |  |  |  | |  | |  |  |  |  |
|  |  | |  | | | | |
|  | |
|  |  |  |  |  |  |
|  | |

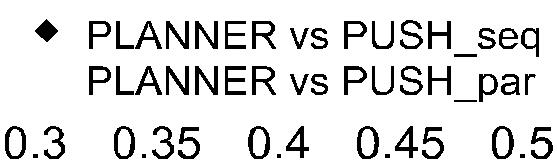
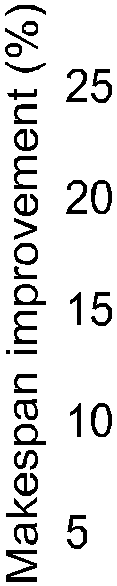


Fig. 8 A dependence of the data production makespan improvement of the proposed model on the bandwidth of the links between remote nodes in the simulations with multiple compu- tational nodes.

the higher CPU utilization. At the same time the opposite transfer paths are used to transfer output files, and that allows to avoid possible bottlenecks. According to the results, the additional usage of an available bandwidth between remote nodes, even as low as 20 Mbps, can result in a significant performance improvement.

The plot at Figure 9 shows the CPU utilization (percentage of busy CPUs over total CPUs in the system) as a function of time for the simulation with 100 Mbps interconnecting links. As it can be observed, both PUSH par and PUSH seq models did not manage to utilize 100 % of CPUs, and the number of busy CPUs is fluctuating over time. This simulation, also, illustrates that the currently used scheduling approach (PUSH par) can lead to significant performance drop when multiple CPUs access the remote data concurrently over a shared bandwidth and, as a result, delay each other. At the same time the PLANNER achieves 100 % CPU utilization shortly after the start and keeps it until the end of simulation due to distributing the network load over the links which would be idle otherwise. The makespan improvement in this simulation is approximately 28 % and 20 % compared to PUSH par and PUSH seq, respectively.

After these simulations we can conclude that the proposed approach can success- fully utilize the available network capacity including indirect links, in order to ”pre- place” the data for computations. This can lead to a significant performance improve- ment comparing to the traditional job scheduling approaches, when data is transferred during the computation and over direct links only.

7 Conclusion

In this paper we proposed a model of distributed data production, where all the files from a single source has to be processed once and transferred back. This model al- lows planning of WAN, storage and CPU loads using the network flow maximization approach. The proposed approach will enable automated and scalable planning and optimization of distributed computations which are highly required in data intensive computational fields such as High Energy and Nuclear Physics. Compared to the cur- rently used data production scheduling the model provides three degrees of optimiza-

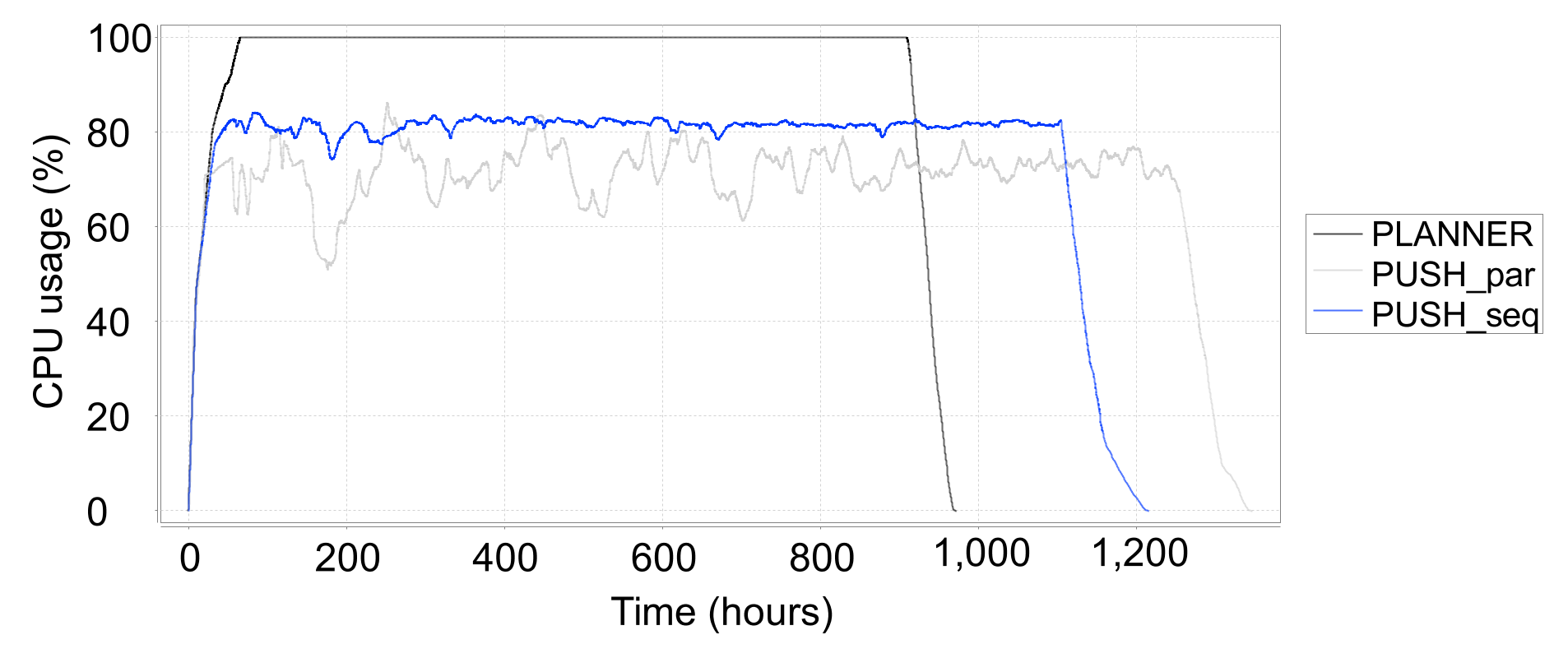


Fig. 9 A global CPU utilization (percentage of busy CPUs over total CPUs in the system) as a function of time for the simulation with 100 Mbps interconnecting links. The Grid structure is given at Figure 7. A set of 60,000 data production jobs was used for the simulations.

tion: transferring input files in advance before their processing which allows to decrease I/O latency; balancing of the network traffic, which includes splitting the load between several alternative transfer paths; and transferring files sequentially in a coordinated manner, which allows to reduce the influence of network bottlenecks.

The model was tested using one of the standard tools for Grid simulation (GridSim). The data extracted from the log database of the real data production framework of the STAR experiment was used as input for simulations. The simulations has shown that the proposed model systematically provides a better performance for distributed computations, which can reach 28 % improvement compared to the current scheduling approach. According to our results, the model can help to efficiently utilize more of the remote computational power using a limited network bandwidth. In addition to that, in our model the jobs are submitted to CPUs only after the data can be accessed efficiently by them, otherwise the CPUs remain idle and can be assigned to other computational jobs. These factors provide more flexibility when setting up a remote data production.

We continue the development of the data production planner, as the next step we plan to verify results of the simulations using data from more HENP experiments (such as ALICE at CERN) and study the effects of a background network traffic. We also plan to study the scalability of the approach, the planner performance and the influence of the parameters, such as planning time interval, on the quality of the solution. The final goal is to integrate the developed planner into the data production framework of the experiment STAR.

Acknowledgements This work has been supported by the Czech Science Foundation (13-

20841S, P202/12/0306), the MEYS grant CZ.1.07/2.3.00/20.0207 of the European Social Fund (ESF) in the Czech Republic: Education for Competitiveness Operational Programme (ECOP) and the Office of Nuclear Physics within the U.S. Department of Energy.

References

1. Ackermann, K.H., et al.: STAR detector overview. Nuclear Instruments and Methods in

Physics Research A499, 624–632 (2003)

2. Ahuja, R.K., Magnati, T.L., Orlin, J.B.: Network flows : theory, algorithms, and applica- tions. Prentice Hall (1993)

3. Balewski, J., Lauret, J., Olson, D., Sakrejda, I., Arkhipkin, D., et al.: Offloading peak processing to virtual farm by STAR experiment at RHIC. Journal of Physics: Conference Series 368(012011) (2012)

4. Beaumont, O., Carter, L., Ferrante, J., Legrand, A., Robert, Y.: Bandwidth-centric al- location of independent tasks on heterogeneous platforms. In: Parallel and Distributed Processing Symposium, Proceedings International, IPDPS. IEEE (2001)

5. Buyya, R., Murshed, M.: 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)

6. Casanova, H., Legrand, A., Zagorodnov, D., Berman, F.: Heuristics for scheduling param- eter sweep applications in grid environments. In: Heterogeneous Computing Workshop,

2000.(HCW 2000) Proceedings. 9th, pp. 349–363. IEEE (2000)

7. Drozdowski, M.: Scheduling for Parallel Processing, 1st edn. Springer-Verlag London

(2009)

8. Faerman, M., Su, A., Wolski, R., Berman, F.: Adaptive performance prediction for dis- tributed data-intensive applications. In: Supercomputing, ACM/IEEE 1999 Conference, pp. 36–36. IEEE (1999)

9. Hajdu, L., Lauret, J., Didenko, L., Amol, J., Betts, W., Jang, H.J., Noh, S.Y.: STAR expe- rience with automated high efficiency Grid based data production framework at KISTI/Ko- rea. In: HEPiX Spring 2015 Workshop. Oxford University, UK (2015)

10. Hwang, K., Dongarra, J., Fox, G.: Distributed and Cloud Computing: From Parallel Pro- cessing to the Internet of Things. Elsevier Science (2013)

11. Klus´aˇcek, D., Rudov´a, H.: Alea 2 – job scheduling simulator. In: Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques (SIMUTools). ICST (2010)

12. Klus´aˇcek, D., Rudov´a, H.: A metaheuristic for optimizing the performance and the fair- ness in job scheduling systems. In: Y. Laalaoui, N. Bouguila (eds.) Artificial Intelligence Applications in Information and Communication Technologies, Studies in Computational Intelligence, vol. 607, pp. 3–29. Springer International Publishing (2015)

13. Magoules, F., Nguyen, T.M.H., Yu, L.: Grid Resource Management: Towards Virtual and

Services Compliant Grid Computing, 1st edn. CRC Press, Inc. (2008)

14. Makatun, D., Lauret, J., Rudov´a, H., Sˇumbera, M.: Model for planning of distributed data production. In: Proceedings of the 7th Multidisciplinary International Scheduling Conference (MISTA), pp. 699–703 (2015)

15. Makatun, D., Lauret, J., Rudova, H., Sˇumbera, M.: Planning for distributed workflows: constraint-based coscheduling of computational jobs and data placement in distributed environments. Journal of Physics: Conference Series 608, 012,028 (2015)

16. Ranganathan, K., Foster, I.: Decoupling computation and data scheduling in distributed data-intensive applications. 11th IEEE International Symposium on High Performance Distributed Computing pp. 352–358 (2002)

17. Santos-Neto, E., Cirne, W., Brasileiro, F., Lima, A.: Exploiting replication and data reuse to efficiently schedule data-intensive applications on grids. In: Proceedings of the 10th International Conference on Job Scheduling Strategies for Parallel Processing, pp. 210–

232. Springer-Verlag, Berlin, Heidelberg (2005)

18. Sinnen, O.: Task Scheduling for Parallel Systems. Wiley Series on Parallel and Distributed

Computing. Wiley (2007)

19. Srinivasan, S., Kettimuthu, R., Subramani, V., Sadayappan, P.: Selective reservation strategies for backfill job scheduling. In: Job Scheduling Strategies for Parallel Processing, pp. 55–71. Springer (2002)

20. Sulistio, A., Poduval, G., Buyya, R., Tham, C.K.: Constructing a grid simulation with differentiated network service using GridSim. In: Proceedings of the 6th International Conference on Internet Computing (2005)

21. TORQUE resource manager. [http://www.adaptivecomputing.com/products/op](http://www.adaptivecomputing.com/products/open-)en- source/torque/

22. Zerola, M., Lauret, J., Bartak, R., Sˇumbera, M.: One click dataset transfer: toward efficient coupling of distributed storage resources and CPUs. Journal of Physics: Conference Series

368(012022) (2012)

23. Disk pool manager (DPM). https://svnweb.cern.ch/trac/lcgdm/wiki/Dpm

24. Korea institute of science and technology information KISTI. [h](http://en.kisti.re.kr/)ttp://en.kisti.re.kr/

25. Portable batch system (PBS). [http://www.pbsw](http://www.pbsworks.com/)orks.com/

26. Xrootd. [http://xro](http://xrootd.slac.stanford.edu/)otd.slac.stanford.edu/

27. JGraphT a free java graph library that provides mathematical graph-theory objects and algorithms. online (2015). URL [http://jgraph](http://jgrapht.org/)t.org/

28. MonAlisa: Grid Online monitoring data of the ALICE experiment. online (2015). URL

[http://alimonitor.cern.c](http://alimonitor.cern.ch/)h/