Network Flows for Data Distribution and Computation

Dzmitry Makatun
Nuclear Physics Institute of the CAS,
Czech Republic
email: dzmitry.makatun@fjfi.cvut.cz

Jérôme Lauret
STAR,
Brookhaven National Laboratory, USA
email: jlauret@bnl.gov

Hana Rudová
Faculty of Informatics, Masaryk University,
Czech Republic
email: hanka@fi.muni.cz

Michal Šumbera
Nuclear Physics Institute of the CAS, Czech Republic
email: sumbera@ujf.cas.cz

***Abstract*—An important class of modern big data applications is distributed data production in High Energy and Nuclear Physics (HENP). Such data intensive computations heavily rely on geographically distributed resources featuring hundreds of thousands CPUs and petabytes of storage. Unfortunately, classical job scheduling approaches either do not address all the aspects of the case or do not scale appropriately. Previously we have developed a new job scheduling approach dedicated to distributed data production, where the load balancing across sites is provided by forwarding data in peer-to-peer manner, but guided by a centrally created and periodically updated plan, aiming to achieve global optimality. Because the many HENP experiments utilize distributed storage, in this work we provide an important generalization of our approach to consider multiple sources of input data. The underlying network flow model is also extended to enable optimization on various additional criteria on top of the flow maximization making it versatile for a wide scope of potential use cases. In this study such additional optimization was used for more efficient reasoning with multiple data sources: balancing their usage and planning of the initial data distribution. Those two considerations allow to reduce an influence of network bottlenecks at early and late stages of data production. The simulations carried out in this work allow to test our approach towards a more general case of networks and servers not limited to specifics of HENP infrastructure. In all of the simulations our planner has shown a significant improvement in both average throughput and makespan against the typically used pull scheduling approach.**

I. Introduction

Distributed data processing in High Energy and Nuclear Physics (HENP) is a prominent example of big data analysis [1]. When applied to petabytes of data being processed at tens of computational sites with thousands of CPUs, standard job scheduling approaches either do not satisfy the problem complexity or are dedicated to one specific aspect of the problem only (CPU, network or storage). As a result, the general orchestration of the system is left to the production managers and requires reconsideration each time new resources are added or withdrawn.

Data production (or pre-processing in big data terminology) is an important type of computations in HENP when each file in a given set has to be processed once. Similar processing scenarios can also be found in other fields. Typically, the outcome of data production can be further exploited only after the entire dataset is processed. Therefore it is highly desirable to achieve as short makespan as possible with available resources. Given the data-intensive nature of data production the network latency often becomes the major limiting factor for overall performance of the distributed computations.

We have addressed this problem in previous research [2], [3] where we considered general properties of data production and developed a new job scheduling approach. In our approach the load balancing across sites is provided by forwarding data in peer-to-peer manner, but guided by a centrally created (and periodically updated) plan, aiming to achieve global optimality. The planner considers network and CPU performance as well as available storage space at each site and plans data movements between them in order to maximize an overall processing throughput. In the later work [4] we have tested our approach in more realistic simulations including background network traffic and computing infrastructure of one of the largest HENP experiments. The planner has shown a significant performance improvement and solving time which allows for online planning in real environment. In this study we extend the scheduling approach to a more general case — when data initially reside (or can be placed) at multiple locations. The recent update to the underlying model has enabled additional optimization on secondary targets, which makes our approach even more versatile. Balancing the usage of data sources and initial data distribution are discussed in this paper as first examples of the secondary optimization. More than that, the novel application of network flows with costs allows consideration of a wide range of scheduling issues as discussed in Conclusion. Our new simulations were focused on large-scale networks of relatively small computing facilities. Such setup of simulations helps to test the planner against more general applications outside of the dedicated infrastructure of HENP computing. It also follows the recent tendency in HENP where the fraction of computations performed at smaller facilities (called Tier-2 sites) is growing comparing to that of the major computational sites (called Tier-1) [5].

A. Related work

Common job scheduling policies [6], such as First Come First Served (FCFS), conservative backfilling (CONS), aggressive backfilling (EASY), selective backfilling, *etc*. consider general types of workloads with jobs of various duration and CPU number. However, none of those algorithms consider network scheduling.

The general methods for scheduling parallel jobs with communication delays are extensively described in [7] and [8]. In [9] an optimization of network latency was achieved by replication of highly used files to more sites while the jobs are executed where their input data are located. Similarly, the Storage Affinity [10] approach exploits data re-utilization to improve the performance of the application in Grid. In contrast to that, data reuse is impossible for the data production problem, since each file has to be processed once. In [11], the authors consider job scheduling on heterogeneous resources (Grid) taking data transfer overhead for each job into account. The input transfer overhead was estimated knowing an end-to-end connection speed, but neither the file transfers were scheduled at network links, nor actual network topology was taken into account. In case of the data-intensive applications, uncoordinated data transfers may oversaturate the network capacity which leads to an overall degraded performance. For this reason in our research we consider planning of network flows.

Modern HENP experiments utilize different types of distributed data management systems (DDM) such as DPM [12], XrootD [13], Hadoop [14] and Ceph [15]. Those systems feature different techniques to balance load across storages and decrease access latency. But DDM's reasoning is disconnected from CPU allocation policy. In other words, it does not consider data placement or movement with respect to the distribution of CPU power or network structure. Hence, it does not provide a possibility to improve the data availability prior to computations, unless it is done by a custom setup such as at [16]. Therefore, one of the main ideas of our planer is to bring the data closer (in a network access sense) to the processing sites by the time when they are needed.

The idea of job forwarding between resources was introduced in [17] to achieve a balanced CPU load in Grid. The forwarding is performed by independent "intelligent agents" at each resource and jobs are sent to the closest free resources. Neither network bandwidth and topology nor transfer latency are considered. Compared to our approach, the idea also lacks an orchestration for global optimization.

An idea of advanced bandwidth reservation at network links has received its development in [18]. The reservation allows to avoid transfer collisions and decrease data access latency. Current works are focused on implementation of such reservations and their scheduling rather than on identifying when they are needed. Our approach can be naturally extended with the bandwidth reservation models, because it provides plans for upcoming data transfers. Study of such extension is planned for future.

II. Problem characteristics

We consider data intensive computations on a set of distributed resources. We focus on a specific type of data processing called data production which is typical for computations in HENP. Our goal is to maximize computational throughput of such data processing. In this section we will describe elements and important aspects of the considered problem.

In HENP experiments raw data produced by a detector and filtered by a trigger system are stored in form of separate files (input files) typically of several gigabytes of size. The aim of data production is to process raw data in order to reconstruct physical events. Each unique input file (containing raw data) creates a unique output file (containing reconstructed data) after processing. The data production is executed by campaigns where a given large dataset (containing petabytes of data) has to be processed on a given set of facilities (containing tens of thousand of CPUs) within an expected time window (several months).

The data production in HENP has a data level of parallelism, which means that it is divided into independent *computational jobs* applying the same processing on different files. For simplicity we assume that each computational job has its unique input file, uses a single CPU and produces a unique output file. Scenarios which have multiple input/output files per job still comply with our model if files of a single job can be grouped for transfer and storage. The size of the input file of a job j is denoted as $InSize_j$, the size of the output file is $OutSize_j$ and the job duration is p_j. 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: $p_j \approx \alpha_i \cdot InSize_j$ and $OutSize_j \approx \beta \cdot InSize_j$, where i is an id of a processing site, α_i and β are considered as constant coefficients. Since all the jobs perform the similar type of processing, knowing an average values of α_i and β for previously completed jobs we can estimate the size of the output and the duration of a particular job.

The computational resources are organized into sites. Each site $c_i \in C$ is a set of closely connected machines with CPUs (also can be referred as a computing facility, cluster or node) 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 site. The key principle to identify a set of machines as a site is that they can access a given local disk with a latency which is negligible compared to their access to outer storages. We assume that there is a local job scheduling system at each site which allows to submit jobs to its machines. 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

site c_i is described by the available local disk space $Disk_i$, the number of CPUs $NCPU_i$ and average processing time per CPU and unit of data α_i. There is an *input queue* of input files which are stored at this site, but have not been processed yet. The input files from this queue can be submitted for processing or transferred to another site if needed. Similarly, output files of previously finished jobs are organized into an *output queue*.

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 mutually interconnected with network links $l \in L$ of known bandwidth $b(l)$. They form a computational network (Grid) which can be described by a directed weighted graph. Such graph can represent a realistic network topology, where the network routers can be considered as sites with no CPUs and local disk and are not assigned any function in data production. In our model we consider only the network latency, which can be calculated as a size of transferred data divided by the bandwidth of the link.

As mentioned before, the main goal is to maximize the overall computational performance for data production. In a real environment the performance can be evaluated using such parameters as a total CPU utilization (given that there is no job duplication and I/O waiting CPU cycles are not counted as useful work) and makespan for processing of a given set of input files, i.e. the completion time of the last output file transfer.

III. Maximum flow approach

In our original as well as proposed approach [2], [3] we do planning by cycles, i.e. a plan for a limited time interval ΔT is created at the beginning of this interval and then the procedure is repeated for the next interval until all the data are processed and all output is transferred. The plan relies on the current system state and recent statistics but not on previously issued plans. Consideration of a shorter interval (e.g. 12 hours) allows to adapt to dynamic environment implying background network load, addition/withdrawal/failure of resources and fluctuation of job parameters. Due to such dynamics a single plan created for an entire production campaign would not remain feasible for its duration (e.g. 1 month). The model discussed in this section is a direct extension of [2], [3] with addition of multiple sources.

For a single planning cycle let us consider a time interval ΔT at an arbitrary moment of data production. We assume that at the beginning of this interval some of the CPUs in the system are busy, and there can be some amount of input data already placed at each processing site. We need to transfer the next portion of input data to each site during this time interval in order to avoid draining of the local input queues. We also need to deliver the output data to its destination as fast as possible.

The computational Grid is represented by a directed weighted graph where vertexes are computational sites $c_i \in C$ and network routers; edges $l \in L$ are network links. We will give two separate problem formulations based on network flows: for input and output transfer planning. In order to formulate both problems we have to define a capacitated $\{s, t\}$ network [19], which is a set of vertices 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 $f(e)$ to each edge $e \in E$ can be found in polynomial time with known algorithms. Here $f(e)$ is understood as an actual amount of data to be transferred within ΔT.

We prioritize transfer of output files, because it allows to create 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.

A. 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. 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 at Fig. 1a.

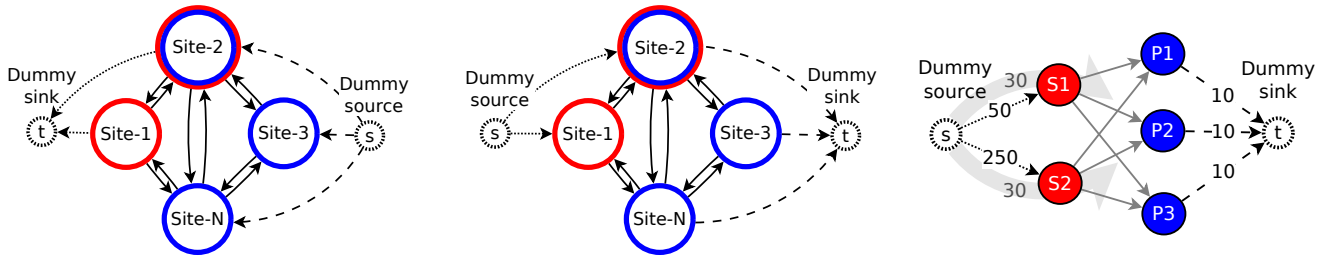
The following expression summarizes the capacities of edges in the output problem:

$$cap(e) = \begin{cases} b(e) \cdot \Delta T & \text{if } e \in L \\ \overline{w}_i & \text{if } e = \overline{d}_i \in \overline{D} \\ \overline{k}_i & \text{if } e = \overline{q}_i \in \overline{Q} \end{cases} \quad (1)$$

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 ΔT.

B. Input flow planning

For the input problem (see Fig. 1b) we apply a similar transformation to the initial graph of the computational network.



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

(b) *Input* flow planning. Solid lines are network links L, 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).

(c) Multiple solutions of the maximum flow problem. S1,S2 are input sources, P1 – P3 are processing sites. The dash and dotted lines are dummy edges, labels show their capacity, solid lines are network links. The grey arrows in the background show two alternative flows through the input sources.

Fig. 1: A computational network represented as a capacitated $\{s,t\}$ graph for dataflow planning.

We add dummy edges $d_i \in D$ from each processing site to the sink, and a 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:

$$cap(e) = \begin{cases} b(e) \cdot \Delta T - f^{out}(e) & \text{if } e \in L \\ w_i & \text{if } e = d_i \in D \\ k_i & \text{if } e = q_i \in Q \end{cases} \quad (2)$$

where w_i is the demand for new input data at processing site c_i. Its calculation is explained in the next Section. 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 l during ΔT.

C. Capacities of dummy edges

In order to complete the formulation of both input and output problems we have to set the values w_i and $\overline{w}_i$. Those values correspond to the estimated amount of input/output data to be processed/produced at processing sites during ΔT. If the system has reached a stable throughput, then we can expect that each site will consume/produce the same amount of data as in the previous planning cycle. For this reason, in most of the cases these values can be derived from the statistics of previous performance. Otherwise, if the system is in a transient state (the start/end of data production or addition/withdrawal of resources) or problems are identified (insufficient input data or free space at some sites) the values w_i and $\overline{w}_i$ can be estimated knowing the current state of the influenced sites. It is important to note, that the value of w_i is not necessary equal to the maximal amount of input data that the site can accept. For example, a site with large storage but few CPUs should not accumulate excessive data instead of processing it at another site. To find a correct estimation we should consider an average computational throughput ($\frac{NCPU_i}{\alpha_i}$), free local disc space and the amount of each type of data placed at the site. The detailed procedure for estimation of w_i and $\overline{w}_i$ can be found in [3].

D. Plan execution

In order to execute the plan, there is a dedicated service running at each site, called *handler*, also responsible for sending statistics and status information to the planner. As the handler receives a plan, it knows the amount of input/output data ($f^{in}(l)$, $f^{out}(l)$) to be transferred over each outgoing link to the neighboring nodes. Each time a new input file arrives over one of the incoming links, the planner either submits it for processing (if there are free CPUs) or, otherwise, forwards it over one of the outgoing links with a remaining flow $f^{in}(l) > 0$ and then decreases this flow by the size of the file. If there are no outgoing links with $f^{in}(l) > 0$ the file is kept at the local queue until a CPU becomes free or a new plan is issued. In this manner the handler maintains a local queue of input files, so that a CPU can start the next job as soon as a previous one is completed, therefore, a transfer overhead is eliminated. The output files are forwarded in a similar way until they reach one of the output destinations. The detailed description of the handler's logic can be found in [3].

IV. Planning with multiple 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 at Fig. 1c. 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 maximal flow on any additional criteria [19]. When an input source gets depleted, the number of possible transfer paths decreases and it potentially leads to emergence of additional bottlenecks. For this reason, a solution taking into account different amount of data at particular sources to properly balance their usage during computation

is needed. 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, an 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 f for a given graph is defined as $\sum_{e \in E} f(e) \cdot cost(e)$. 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 generalized push-relabel algorithm [20] 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 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 are changing as they are depleted.

V. Initial data distribution

Another important question concerning multiple input sources is: what would be the best way to distribute the given dataset over available storages prior to computations? A carefully planned initial data distribution can help to reduce the subsequent data production makespan. In 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 [5] where raw data are persistently stored at 12 geographically separated sites and is reprocessed (typically several times during years) using updated algorithms and calibration data in order to improve quality of the resulting reconstructed data. Such re-processings 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 distribute the raw data in order to decrease the makespan of upcoming data (re)processing.

The general idea of the initial planning stage is to consider the entire data production in one planning cycle where ΔT equals to 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.

A. Model description

Let K 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 $S_i > 0$. 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 use the min-cost max-flow approach to the initial data distribution problem. Similar as before, 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 S_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 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 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 amount 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.

B. 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 K with a predefined precision ε. The overall solving procedure for the initial planning problem is the following:

1) Calculate an optimistic (as if there is no network bottlenecks) estimation for makespan $\Delta T = K / \sum \frac{NCPU_i}{\alpha_i}$,
2) Construct the $\{s, t\}$ network as described before,

3) Update capacities of the edges using the current value of ΔT
4) Solve the max-flow min-cost problem. If $\mid K - \Phi \mid < K \cdot \varepsilon$ then go to (6),
5) Set a new value for the estimated makespan $\Delta T = \frac{K}{\Phi} \cdot \Delta T_{previous}$ and go to (3) ,
6) Resulting flows f over dummy edges q_i are the amount of data to be placed at each source ($k_i = f(q_i)$), 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.

VI. Experiments

For testing of our updated scheduling approach, we have performed simulations of distributed data production using GridSim [22] — a common tool for simulation of computational Grids. Network links were modeled as *space-shared* resources, i.e. if multiple files are submitted for a transfer over the same link simultaneously, they are dispatched one by one in the First-In-First-Out order. We have used job parameters from log records collected during the data production for the STAR experiment in June – September 2014 at the KISTI computational facility [23]. The records of 60,000 of jobs were collected which corresponds to 260 TB of input files processed at 1,000 CPUs.

The planner was implemented in Java using the JGraphT [24] library which provides common tools for graphs and flows. The simulations were running under Windows 10 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. An average runtime of simulations described in this paper was 6 minutes with deviation 5 minutes.

A. PULL scheduling approach

In order to compare our approach with others we have simulated job scheduling typical for modern distributed computing systems and HENP computations in particular. Most of the scheduling systems in HENP experiments have their own implementation of so called *pull model*. While details of implementations may differ, the general pattern is very similar: a *pilot job* [5] 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 node
- 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 current source is depleted, switch to the next in the queue. Repeat until all data are processed

Output files can be send to a single storage, or to multiple ones, using the same reasoning as for input files. In the following text the above algorithm is called PULL for brevity. Our scheduling approach is referred as PLANNER.

B. Experiment setup

In previous work we have tested our approach in simulations of data production in the network of so-called Tier-1 sites of one of the largest HENP experiments. Eleven large sites with $NCPU_i$ varying from 500 to 12,000 interconnected with a dedicated network were considered in that simulation [4]. In this work we focus on a large-scale network consisting of several tens of smaller sites (referred as Tier-2) interconnected with relatively slow links. This provides an opportunity to test our planner over a wider scope of infrastructures and consider its application beyond the HENP computations including commodity networks and computational resources.

It is known that large-scale communication networks (including the Internet) reveal properties of a scale-free graph [25]. For this reason the network topology in our simulations was constructed using such a graph. The parameters of the computational sites (CPU, disk space) were set to comply with observations from the online monitoring data of HENP experiments (such as MonAlisa [26]). The bandwidth of the links was set considering the lower bound required to utilize CPUs efficiently, which is close to 250 kbps per CPU according to our previous simulations [3]. This allows to test if the simulated scheduling approaches can utilize the bounded bandwidth efficiently. We used the following procedure for grid generation:

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

Ten generated computational networks used in the simulations consisted of 50 sites each and were varying in total number of CPU's (4,000 – 23,000) and bandwidth of the links. Fig. 2 shows an example of such network with 50 sites, 8 input sources, 77 links and 6,663 CPUs.

Four simulations were done for each generated network: PULL and PLANNER using multiple input sources; and

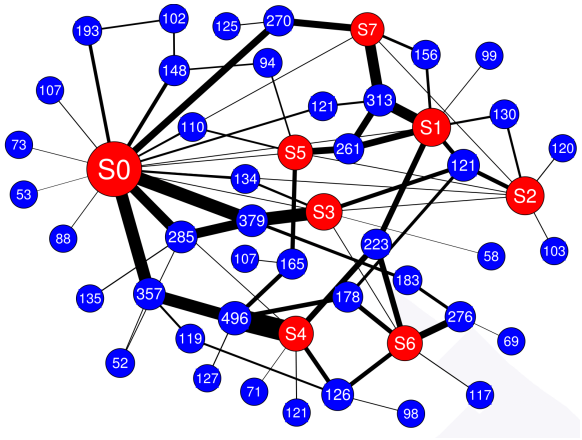


Fig. 2: Example of a computational network generated for simulations. The red vertexes are input sources, the blue vertexes are processing sites, where the label is the number of CPUs. The thickness of the links illustrates the bandwidth.

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 Sec. V. In case of a single source, the site with the best connectivity (largest degree) was used. S0 site at Fig. 2 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 was submitted for data production and the resulting makespan and CPU utilization over time were compared. The ΔT for the planner (from Sec. IV) was set to 12 hours.

C. Results

In all the simulations the PLANNER has reached the highest CPU utilization (both peak and average) and a significantly shorter makespan. A typical dependence of an overall CPU usage over time is presented at Fig. 3. 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 at the plot. It is a well known drawback of remote data access in scale-free networks. Such behavior 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 send to distant (in a network sense) processing sites, especially at the end of 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 60 % in this particular simulation and by 46 % on average with deviation 12 % in all the simulations (PLANNER compared to PULL(single), note that PULL(single) has better performance than PULL as discussed later). Such a significant makespan

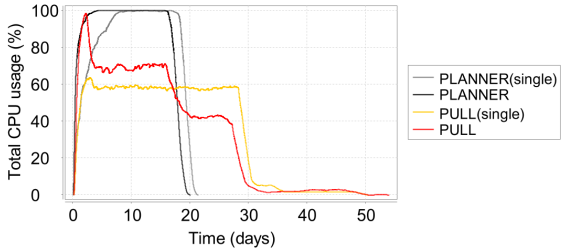


Fig. 3: Total CPU usage in simulated data production using the network pictured at Fig. 2.

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) at Fig. 3) and thereby provides an additional decrease of the makespan by 6 % on average with deviation 3 % compared to usage of 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. 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 plan to consider data replication in our future work.

Simulations of our planner using a single input source can be compared to simulations in our previous papers [3], [4], where we have observed up to 28 % makespan improvement in networks with several sites. Comparing it to the results of the recent simulations we can observe that the gain in the makespan grows with the complexity of the network. The makespan improvement of PLANNER(single) against PULL(single) has reached 43 % on average with deviation 10 % in the recent simulations.

VII. Conclusion

Our job scheduling approach is dedicated to data-intensive applications on distributed resources when network performance and storage space are important factors along with CPU throughput. In this work we have generalized the network flow approach towards reasoning over multiple criteria. Given that we have extended our approach to consider multiple sources of input data. To enable efficient utilization of many sources

additional reasoning was introduced to the underlying model: planning of initial data distribution and balancing usage of the sources.

While initially focused on distributed data production in HENP, our approach can also be helpful in other applications where a large set of data at distributed storage has to be processed on geographically spread resources within the shortest possible time. The approach is especially beneficial when the network performance becomes a limiting factor and optimization of data access is required. It can also address usage of volatile cloud resources provided on demand due to its adaptability to changing environment.

In complement to the simulations performed previously, in this paper we have tested our approach in realistic large-scale networks with tens of smaller computational sites. The simulated environment follows the recent trend in HENP computations: the aggregated computational power of smaller sites (Tier-2) is growing faster than that of the larger sites (Tier-1). These simulations also allowed to study the behavior of our approach outside of the dedicated infrastructure of HENP experiments and consider more general case of networks and servers.

In all of the simulations our planner has shown a significant improvement against the typically used pull scheduling approach. The peak processing throughput was increased and the makespan of data production was improved by 46 %. The planning of initial data distribution across several sites allowed to reach peak throughput faster and to decrease the makepan by up to 18 % comparing to the case when a single input source is used.

Our new extension of the planner's mathematical model is very important for many close scheduling problems because the cost function can be used to handle various optimization criteria on top of the flow maximization. Such extension makes our planner versatile to many potential use cases. This paper proposes two ways of using the cost: to balance source usage and to plan initial data distribution. In future it can also be used to impose other desired properties to plans. For example, it allows to prioritize the usage of faster CPU resources, minimize expenses for using external paid cloud resources or decrease the load on shared network links experiencing high background traffic and other. We plan to study and implement such additional optimization in future.

Acknowledgment

This work has been supported by the Czech Science Foundation 13-20841S, the MEYS grant LG15001 and the Office of Nuclear Physics within the U.S. Department of Energy.

References

[1] H. Hu, Y. Wen, T.-S. Chua, and X. Li, "Toward scalable systems for big data analytics: A technology tutorial," *IEEE Access*, vol. 2, pp. 652–687, 2014.

[2] D. Makatun, J. Lauret, H. Rudová, and M. Šumbera, "Model for planning of distributed data production," in *Proceedings of the 7th Multidisciplinary International Scheduling Conference (MISTA)*, 2015, pp. 699–703.

[3] ——, "Distributed data production planning for High Energy and Nuclear Physics," 2016, (Under submission process).

[4] ——, "Multi-resource planning: Simulations and study of a new scheduling approach for distributed data production in high energy and nuclear physics," *Journal of Physics: Conference Series*, 2016, (Accepted for publication).

[5] I. Bird, P. Buncic, F. Carminati, M. Cattaneo, P. Clarke, I. Fisk, M. Girone, J. Harvey, B. Kersevan, P. Mato, R. Mount, and B. Panzer-Steindel, "Update of the Computing Models of the WLCG and the LHC Experiments," CERN, Geneva, Tech. Rep. CERN-LHCC-2014-014. LCG-TDR-002, Apr 2014.

[6] D. G. Feitelson and A. M. Weil, "Utilization and predictability in scheduling the IBM SP2 with backfilling," in *Proceedings of the 12th International Parallel Processing Symposium IPPS/SPDP*. IEEE, 1998, pp. 542–546.

[7] M. Drozdowski, *Scheduling for Parallel Processing*, 1st ed. Springer-Verlag London, 2009.

[8] O. Sinnen, *Task Scheduling for Parallel Systems*, ser. Wiley Series on Parallel and Distributed Computing. Wiley, 2007.

[9] K. Ranganathan and I. Foster, "Decoupling computation and data scheduling in distributed data-intensive applications," *11th IEEE International Symposium on High Performance Distributed Computing*, pp. 352–358, 2002.

[10] E. Santos-Neto, W. Cirne, F. Brasileiro, and A. Lima, "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*. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 210–232.

[11] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman, "Heuristics for scheduling parameter sweep applications in grid environments," in *Proceedings of the 9th Heterogeneous Computing Workshop*. IEEE, 2000, pp. 349–363.

[12] "Disk pool manager (DPM)," https://svnweb.cern.ch/trac/lcgdm/wiki/Dpm.

[13] "Xrootd," http://xrootd.slac.stanford.edu/.

[14] T. White, *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

[15] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proceedings of the 7th symposium on Operating systems design and implementation*. USENIX Association, 2006, pp. 307–320.

[16] J. Balewski, J. Lauret, D. Olson, I. Sakrejda, D. Arkhipkin *et al.*, "Offloading peak processing to virtual farm by STAR experiment at RHIC," *Journal of Physics: Conference Series*, vol. 368, no. 012011, 2012.

[17] J. Cao, D. P. Spooner, S. A. Jarvis, and G. R. Nudd, "Grid load balancing using intelligent agents," *Future generation computer systems*, vol. 21, no. 1, pp. 135–149, 2005.

[18] I. Foster, A. Roy, and V. Sander, "A quality of service architecture that combines resource reservation and application adaptation," in *Proceedings of the 8th International Workshop on Quality of Service*. IEEE, 2000, pp. 181–188.

[19] R. K. Ahuja, T. L. Magnati, and J. B. Orlin, *Network flows: theory, algorithms, and applications*. Prentice Hall, 1993.

[20] A. V. Goldberg, "An efficient implementation of a scaling minimum-cost flow algorithm," *Journal of Algorithms*, vol. 22, no. 1, pp. 1 – 29, 1997.

[21] L. Hajdu, J. Lauret, L. Didenko, J. Amol, W. Betts, H. J. Jang, and S. Y. Noh, "STAR experience with automated high efficiency Grid based data production framework at KISTI/Korea," in *HEPiX Spring 2015 Workshop*. Oxford University, UK, 2015.

[22] R. Buyya and M. 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)*, vol. 14, 2002.

[23] L. Hajdu, L. Didenko, J. Lauret, J. Amol, W. Betts, H. J. Jang, and S. Y. Noh, "Automated finite state workflow for distributed data production," in *ACAT*, 2016, (Submitted for publication).

[24] "JGraphT a free java graph library that provides mathematical graph-theory objects and algorithms." August 2015. [Online]. Available: http://jgrapht.org/

[25] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the internet topology," in *ACM SIGCOMM computer communication review*, vol. 29, no. 4. ACM, 1999, pp. 251–262.

[26] "MonAlisa: Grid online monitoring data of the ALICE experiment," http://alimonitor.cern.ch/, 2015.