Distributed data processing for High Energy Physics

Dzmitry Makatun

3rd year of PGS, email: dzmitry.makatun@fjfi.cvut.cz
Department of Mathematics
Faculty of Nuclear Sciences and Physical Engineering, CTU in Prague

advisors:

Jérôme Lauret, STAR, Brookhaven National Laboratory, USA

Hana Rudová, Masaryk University, Czech Republic

Michal Šumbera , Nuclear Physics Institute, Academy of Sciences, Czech Republic

Abstract.

Keywords: constraint programming, Grid, Cloud, data processing , data transferring, data production, planning, scheduling, optimization, computational jobs, batch system.

Abstrakt.

Klíčová slova:

Contents

1 Introduction

The STAR experiment 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 intends to rely on distributed data processing, making use of several remote computational sites (for some experiments this number can scale up to several hundreds).

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 computation performance and can reduce the CPU time / wall time ratio due to excessive I/O wait. Widely used data management systems in HENP community (Xrootd, DPM) are focused on providing heterogeneous access to distributed storage and do not consider data pre-placement with respect to available CPUs, job durations or network performance. At the same time job scheduling systems (PBS, Condor) do not reason about transfer overheads when accessing data at distributed storage. For this reason, an optimization of data transferring and distribution across multiple sites is often done manually, using a custom setup for each particular infrastructure [2].

1.1 Data production

1.2 Previous work

Previous collaborative work between BNL (Brookhaven National Laboratory) and NPI/ASCR (Nuclear Physics Institute, Academy of Sciences of the Czech Republic) showed that the global planning of data transfers within the Grid can outperform widely used heuristics such as Peer-to-Peer and Fastest link (used in Xrootd)[6, **?**]. Those results became the ground for continuation of research and extension of global planning to the entire data processing workflow, i.e., scheduling of CPU allocation, data transferring and placement at storage.

In previous, we addressed the problem of efficient data transferring in a Grid environment [6]. Data transfers between n computational sites and m data locations were considered but job scheduling was not covered by that work. In [4] 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 planing of resource load can be completed in the first stage before scheduling file transfers

and jobs. In this work 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.

2 Problem analysis

Long I/O overheads when accessing data from remote site can significantly reduce the application's CPUtime/WallTime ratio [**?**, **?**]. For this reason, when setting up a data production at remote sites one has to consider the network throughput, available storage and CPU slots. When there are few remote sites involved in the data processing, the load can be tuned manually and simple heuristic may work, but, as the number of sites grows and the environment is constantly changing (site outage, fluctuations of network throughput and CPU availability), an automated planning of workflows becomes needed.

As an intuitive example of optimization let us consider a situation when a given dataset can be either processed locally, or can be sent to a remote site. Depending on transfer overhead it may appear to be optimal to wait for free CPU slots at the local site and process all the data there, or send a smaller fraction of the dataset for remote processing. Commonly used heuristics such as "*Pull a job when a CPU slot is free*" will not provide an optimization with respect to an overall processing makespan.

Another example arises from a workflow optimization which was done for inclusion of the ANL (Argonne National Laboratory) computational facility into the Cloud based data production of the STAR experiment [**?**]. In this case, and due to the lack of local storage at the site for buffering, the throughput of a needed direct on-demand network connection between BNL (New York) and ANL (Illinois) was not sufficient to saturate all the available CPUs at the remote site. An optimization was achieved by feeding CPUs at ANL from two sources: directly from BNL and through an intermediate site NERSC (National Energy Research Scientific Computing Center, California) having large local caching and with better connectivity to ANL. This example illustrates an efficient use of indirect data transfers which cannot be guessed by simple heuristics. A general illustration of distributed resources used for data production and their interconnection is given at Figure 1.

Scheduling of computational jobs submitted by users (user analysis) has even more degrees of possible optimization: selection between multiple data sources, grouping of jobs that use the same input files. This case becomes even more complex due to a poor predictability of the user analysis jobs. However, the main question for optimization remains the same as for the examples above: How to distribute a given set of tasks over the available set of resources in order to complete all the tasks within minimal time?

2.1 Existing solutions

2.2 Related works

Optimization of data intensive applications in Grid was studied in [5]. In this work an optimization was achieved by replication of highly used files to more sites while the jobs

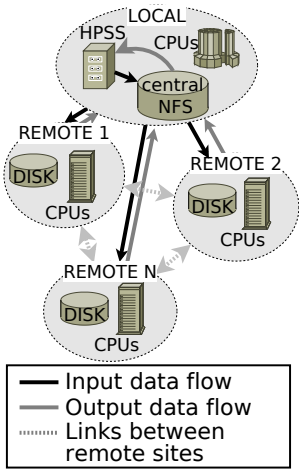

Figure 1: Schema of data production in the Cloud.

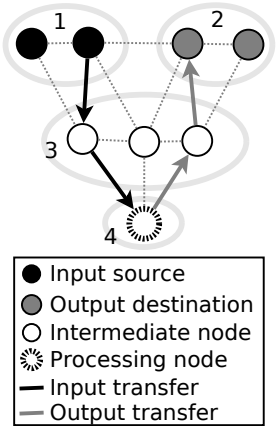

Figure 2: An example of a transfer path. Illustration for constraints 1-4 in section 3.3.2.

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. Explicit model distributing jobs over a Grid with respect to the network bandwidth was proposed in [3]. 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 parameters.

2.3 Usecases

3 Problem formalization

3.1 Network flow maximization approach

Due to a data level of parallelism a typical workflow of HENP computation consists of independent jobs using one CPU, one input and one output file. We assume there is a local scheduler running at each site, which picks a new input file to process from the local storage of that site each time when a CPU becomes free. Input data must be transferred from the central storage to each site in such a manner that at the every moment of time there is enough input files at each site to keep all the available CPUs busy while not exceeding the local storage and network throughput. Another task is to transfer the output files back to central storage, cleaning each local storage for the new input.

Let us consider a scheduling time interval ΔT. We assume that at the starting moment all the CPUs in the Grid are busy, and there is 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 ΔT in order to avoid draining of the local queue by the end of this interval.

The computational Grid is represented by a directed weighted graph where vertexes $c_i \in C$ are computational nodes and edges $l_j \in L$ are network links. Weight of each link b_j is the amount of data that can be transferred over the link per unit of time (i.e. bandwidth). One of the nodes c_0 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 c_0 from the computational nodes. We will give two separate problem formulations: for an input and output transfer planning.

In order to formulate a network flow maximization problem [1] 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 $e \in E$ with their capacities $cap(e)$. A solution that assigns a nonnegative integer number $f(e)$ to each edge $e \in E$ can be found in polynomial time with known algorithms.

3.1.1 Input flow planning

In order to transform a given graph of a Grid into a capacitated $\{s,t\}$ network for an input transfer problem we add two dummy vertexes: a source s and a sink t. Next we add dummy edges $d_i \in D$ from each computational node i to the sink, and a dummy edge q_0 from the source s to the central storage c_0. These dummy edges allow us to introduce constraints on the storage capacity of the nodes. The set of vertexes V consists of computational nodes C and dummy vertexes: $V = C \cup \{s,t\}$. 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 q_0: $E = L \cup D \cup \{q_0\}$. Capacity of each edge defines the maximal amount of data that can be transferred over an edge within time interval ΔT:

$$cap(e) = \begin{cases} b_j \cdot \Delta T & \text{if } e = l_j \in L \\ w_i & \text{if } e = d_i \in D \\ k_0 & \text{if } e = q_0 \end{cases} \quad (1)$$

where w_i is the maximal amount of data that can be transferred to the node i without exceeding its storage capacity $Disk_i$ and k_0 is the total size of available input files at c_0. We denote the solution for the input transfer problem as $f^{in}(e)$.

3.1.2 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. In this case the capacity $\overline{k}_0$ of the dummy edge $\overline{q}_0$ leading from the central storage c_0 to the sink s is equal to the amount of data which can be transferred to c_0 within time interval ΔT (it is limited by the available space at the central storage). The capacity $\overline{w}_i$ of dummy edges $\overline{d}_i$ leading from the source t to computational nodes c_i is equal to the maximum amount of output data which can be transferred from the node c_i.

$$cap(e) = \begin{cases} b_j \cdot \Delta T & \text{if } e = l_j \in L \\ \overline{w}_i & \text{if } e = \overline{d}_i \in \overline{D} \\ \overline{k}_0 & \text{if } e = \overline{q}_0 \end{cases} \quad (2)$$

We denote the solution for the output transfer problem as $f^{out}(e)$.

3.1.3 Calculating capacity of dummy edges

Let us consider data production jobs which perform the same type of processing on the same type of files. Duration p_j of job j processing input file of size $InSize_j$ at node i is

$$p_j = \alpha_i \cdot InSize_j \tag{3}$$

Figure 3 shows a distribution of parameter α for 7000 of data production jobs executed at the same computational site (KISTI). It can be observed that the narrow peaks correspond to jobs using input files of the different type (`"st_physics"`, `"st_physics"`, `"st_physics_adc"`, `"st_jet"`, etc.). For this reason, parameter α can be considered constant for a given type of data processing at a given site.

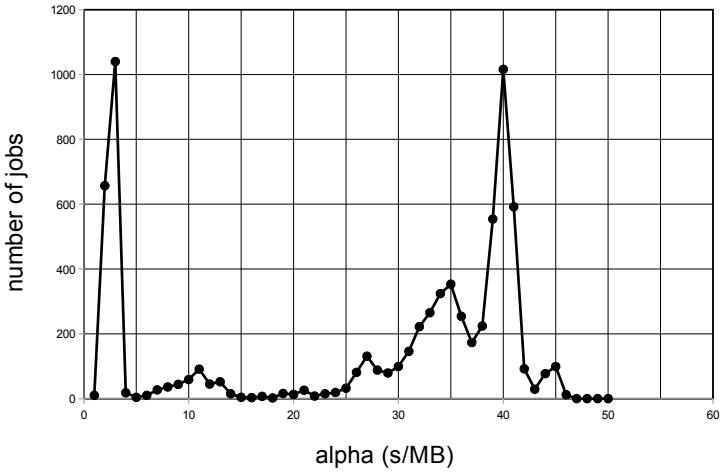

Figure 3: Distribution of parameter α (job duration divided by input file size) calculated for sample of 7000 jobs executed at KISTI during June - September 2014. Narrow peaks correspond to jobs using input files of different types (`"st_physics"` at 40 s/MB, `"st_physics_adc"` at 3 s/MB, `"st_jet"` at 35 s/MB, etc.). For this reason, parameter α can be considered constant for a given type of data processing.

The ratio of size of input $InSize_j$ and output $OutSize_j$ files of each job j is considered to be constant for the same type of data processing:

$$OutSize_j = \beta \cdot InSize_j \tag{4}$$

Figure 4 shows a distribution of parameter β for 7000 of data production jobs executed at the same computational site (KISTI). Similarly to written above, for a given type of data processing this parameter can be considered constant.

During time interval ΔT (which should be long enough) a node i with number of CPUs $NCPU_i$ will process $\frac{1}{\alpha_i} \cdot NCPU_i \cdot \Delta T$ of input data and will produce $\frac{\beta}{\alpha_i} \cdot NCPU_i \cdot \Delta T$ of output data.

Let us consider a storage at a processing node i during time interval ΔT. $Disk_i$ is available disk space at node i. The I_i^{in} is the initial size of input data stored at the local

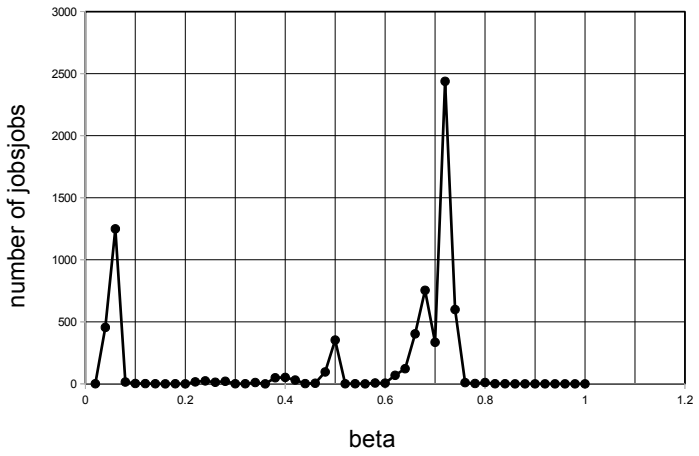

Figure 4: Distribution of parameter β (ratio of output file size over input file size) calculated for sample of 7000 jobs executed at KISTI during June - September 2014. Narrow peaks correspond to jobs using input files of different types (`"st_physics"` at 0.72, `"st_physics_adc"` at 0.04, `"st_fms"` at 0.5 , etc.). For this reason, parameter β can be considered constant for a given type of data processing.

storage; I_i^{out} is the initial size of output data at the storage; New_i^{in} is the amount of input data that will be transferred to i during ΔT; Del_i^{in} is the amount of input data that will be deleted from the storage, because the jobs using these input data will be completed; Del_i^{out} is the amount of output data which will be deleted from the storage, because it will be transferred out of the node; New_i^{out} is the size of new output files to be created during ΔT; Min_i^{in} is minimal amount of input data (this includes input files of running jobs and files in the local queue to ensure stable CPU saturation), Min_i^{out} - total size of output files which can not be transferred because the jobs which produce them are not finished (output files of running jobs).

In the end of ΔT there should be enough input data to keep CPUs busy:

$$I_i^{in} + New_i^{in} - Del_i^{in} \geq Min_i^{in} \geq 0 \tag{5}$$

In the end of ΔT the storage capacity should not be exceeded:

$$0 \leq I_i^{in} + I_i^{out} + New_i^{in} + New_i^{out} - Del_i^{in} - Del_i^{out} \leq Disk_i \tag{6}$$

The balance of the output data at the end of ΔT is:

$$I_i^{out} + New_i^{out} - Del_i^{out} \geq Min_i^{out} \geq 0 \tag{7}$$

If the scheduling interval ΔT is long enough, then we can use the following approximation:

$$Del_i^{in} \approx \frac{1}{\alpha_i} \cdot NCPU_i \cdot \Delta T \tag{8}$$

and

$$New_i^{out} \approx \frac{\beta}{\alpha_i} \cdot NCPU_i \cdot \Delta T \tag{9}$$

Then, combining Equations 5, 6, 8 and 9 we can estimate amount of new input data New_i^{in} which can be transferred to a node.

$$Min_i^{in} + \frac{1}{\alpha_i} \cdot NCPU_i \cdot \Delta T - I_i^{in} \leq New_i^{in} \leq Disk_i - I_i^{in} - I_i^{out} + \frac{1-\beta}{\alpha_i} \cdot NCPU_i \cdot \Delta T + Del_i^{out} \tag{10}$$

Similarly, using Equations 7 and 9 the amount of output data which can be deleted from a node is

$$Del_i^{out} \leq I_i^{out} + \frac{\beta}{\alpha_i} \cdot NCPU_i \cdot \Delta T - Min_i^{out} \tag{11}$$

Since w_i was defined in Subsection **??** as maximal amount of input data that can be transferred to the computing node c_i, we can now define it using Equation 10:

$$w_i = Disk_i - I_i^{in} - I_i^{out} + \frac{1-\beta}{\alpha_i} \cdot NCPU_i \cdot \Delta T + Del_i^{out} \tag{12}$$

where Del_i^{out} is equal to the amount of data which will be transferred from node c_i, i.e. the solution to the output transfer problem $f^{out}(\overline{d}_i)$ (see Subsection **??**). The other values used in Equation 12 can be obtained from monitoring data.

Similarly, $\overline{w}_i$ was defined in Subsection **??** as maximal amount of output data which can be transferred from node c_i. From Equation 11 we obtain:

$$\overline{w}_i = I_i^{out} + \frac{\beta}{\alpha_i} \cdot NCPU_i \cdot \Delta T - Min_i^{out} \tag{13}$$

where ΔT, Min_i^{out} are parameters of the scheduler, and the rest of the values can be extracted from monitoring system.

3.2 Disjoint input and output transfer

In this subsection we will prove that maximum flow problems for input and output transfers can be solved independently under assumptions: (a) all the real network links in considered Grid are full-duplex, i.e. network throughput between two nodes is the same in both directions (b) in stable regime size of output transferred from each node is proportional to size of input transferred to that node at each scheduling interval, i.e. $f^{out}(\overline{d}_i) = \beta \cdot f^{in}(d_i)$, where $\beta \leq 1$.

Let us consider two distinct computational nodes c_1 and c_2 connected by two opposite directed links $l_1 = (c_1, c_2)$ and $l_2 = (c_2, c_1)$ with equal capacities $cap(l_1) = cap(l_2)$. If a solution of the input transfer problem assigns flows to this links such that $f^{in}(l_1) \geq f^{in}(l_2) > 0$ then we can substitute such solution $f^{in}(e)$ with a new one $\widehat{f}^{in}(e)$ where $\widehat{f}^{in}(l_1) = f^{in}(l_1) - f^{in}(l_2)$, $\widehat{f}^{in}(l_2) = 0$ and flows over the rest of the links are unchanged. The same is true for output transfer. This proves that in an optimal solution the same type of files (input or output) are transferred between any two nodes in one direction only, i.e. over one of directed links only.

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

3.2.1 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., $f^{out}(\overline{d}_i) = \beta \cdot f^{in}(d_i)$, where $\beta \leq 1$.

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:

1. Calculate values for $\overline{w}_i$ using Eqn. 13.
2. Solve the problem for output data flows to obtain $f^{out}(e)$.
3. Using Eqn. 12 and $Del_i^{out} = f^{out}(\overline{d}_i)$ calculate w_i.
4. For real links $l \in L$ reduce the capacity by the amount which is used by output transfers: $cap(l_j) = b_j \cdot \Delta T - f^{out}(l_j)$.
5. Solve the problem for input transfers with w_i and $cap(l_j)$ defined in previous steps. Find input data flows $f^{in}(e)$.

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

3.3 Constraint programming approach

Problems of scheduling, planning and optimization are being commonly solved with the help of Constraint Programming (CP) [**?**]. It is a form of declarative programming which is widely used in scheduling, logistics, network planning, vehicle routing, production optimization etc... In the next sections we will introduce our Constraint Satisfaction Problem (CSP) formulation for a data production at multiple sites and provide a simulation-based evaluation of the proposed model.

We will introduce only the core concepts of our CSP formulation and search algorithms, omitting detailed mathematical expressions. The following input parameters are necessary to define our CSP.

Computational Grid (see Figure 1) is described by directed weighted graph where nodes are computational sites c with a given number of CPUs cpu_c and storage space $disk_c$; edges are network links l with weight $slowdown_l$ which is the time required to transfer a unit of data ($slowdown_l = \frac{1}{throughput_l}$). A dedicated storage facility, such as HPSS, can also be modeled as a node with $cpu_c = 0$.

Set of jobs. Each job j has a $duration_j$, it needs one input file of $inputSize_j$, produces one output file of $outputSize_j$, input file is placed at $inputSourceNodes_j$ and output file must be transferred to one of $outputDestinationNodes_j$.

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

Resource selection variables define a node $ProcessingNode_j$ where the job j will be executed and a transfer path for each file f (either input or output of a job). The transfer path is described by a set of boolean variables X_{fl} where *true* means that a file f will be transferred over a link l and $false$ means the opposite.

Time variables are: Js_j is a start time of a job j, Ts_{fl} is a start time of a transfer of a file f over a link l, Fs_{fc} is a start time of a placement of a file f at a node c, $Fdur_{fc}$ is a duration of a placement of a file f at a node c.

In our model we assume that a network link can be modeled as an unary resource with no loss of generality. The measurements in [6] have shown, that a sequential transfer of a set of files does not require more time than a parallel transfer of the same set of files over the same link.

3.3.1 Search overview

We use an incomplete search which can provide a suboptimal solution of required quality within a given time limit because the final goal is to create a planner that can process requests online. For a better search performance the overall problem is divided into two subproblems and the search is performed in two stages:

1. Planning Stage: instantiate a part of variables in order to assign resources for each task.
   (a) Assign jobs to computational nodes.
   (b) Select transfer paths for input and output files.
   (c) Estimate a makespan for a given resource assignment $estMakespan$.
   (d) Find a solution for the subproblem with a minimal estimated makespan.
2. Scheduling stage: define a start time for each operation.
   (a) Define the order of operations.

(b) Put cumulative constraints on resources in order to avoid their oversaturation at any moment of time.

(c) Find a solution with a minimal *makespan* which is the end time of the last task.

3.3.2 Constraints at the planning stage

At the planning stage we have to assign a transfer path for an input and an output file of each job which can be defined by the following constraints (see Figure 2):

1. An input file has to be transferred from one of its sources over exactly one link.

2. An output file has to be transferred to one of its destinations over exactly one link.

3. An intermediate node (neither source, destination nor selected for the job execution) either has exactly one incoming and outgoing transfer or is not on a transfer path: $\exists$ incoming transfer $\Leftrightarrow \exists$ outgoing transfer.

4. There must exist exactly one incoming transfer of an input file and exactly one outgoing transfer of an output file at the node which was selected for the job execution.

5. A file can be transferred from/to each node at most once.

In addition, we use constraints for loop elimination similarly as it is described in [**?**].

3.3.3 Constraints at the scheduling stage

At the scheduling stage the problem is to assign a start time for each task. The following constraints on order of tasks are implemented:

- An outgoing transfer of a file from a node can start only after an incoming transfer to that node is finished. The first transfer of an input file from its source and the first transfer of an output file from the processing node are exceptions from this constraint.
- A job can start only after the input file is transferred to the selected processing node.
- An output file can be transferred only after the job is finished.
- A reservation of space for a file at a node is made when a transfer to that node starts.
- A file can be deleted from the start node of a link after the transfer is finished.
- A reservation of space for an output file is made at the processing node when the job starts.
- An input file can be deleted from a processing node after the job is finished.

Table 1: Variables and parameters used in cumulative constrains on resources.

Task	Start	Duration	Usage	Limit
Job	Js_{jc}	$duration_j$	1	cpu_c
Transfer	Ts_{fl}	$size_f \cdot slowdown_l$	1	1
File placement	Fs_{fc}	$Fdur_{fc}$	$size_f$	$disk_c$

Cumulative constraints are widely used in Constraint Programming for description of resource usage by tasks. Each cumulative constraint requires that a set of tasks given by *start times*, *durations* and *resource usage*, never require more than a *resource limit* at any time. In our case we use three sets of cumulative constraints: for CPUs, storages and links (see Table 1).

4 Simulations

The constraint satisfaction problem was implemented using MiniZinc [**?**] and Gecode [**?**] was used as a solver. The timelimit was set to 3 minutes for both planning and scheduling stages. The simulations were running under Windows 8 64-bit on a computer with Intel i5 (4 cores) 2.50 GHz processor and 6 GB of memory installed. The Gecode solver was running in a parallel mode using 4 threads.

The simulated environment consisted of 3 nodes: a central storage HPSS ($cpu_{HPSS} = 0$) which was the single source for input files and the single destination for output files, a local processing site and a remote processing site. The slowdown of links between the central HPSS and the local site was set to 0, which means that transfer overheads to/from the local site are negligible comparing to a job duration. The slowdown of the links to/from the remote site was increasing in each simulation proportionally to a slowdown factor. The parameters of jobs were taken from logging system of the STAR experiment's data production at computational site KISTI (Korea Institute of Science and Technology Information) [**?**]. The average job duration was 3,000 minutes and average time of transfer was 5 and 10 minutes to/from the remote site respectively (in the simulations where the slowdown factor = 1). Then, in further simulations the transfer times increase proportionally to the slowdown factor. In the simulated environment 80% of CPUs were available at the local site and 20% at the remote site. 2,000 of jobs were scheduled stepwise by subsets (chunks) of 200. Storage constraints were not considered in these simulations. Four different scheduling strategies were compared:

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

Equal CPU load: Jobs are distributed between nodes with the goal to maintain an equal ratio of job duration per CPU. Each input file is transferred prior to the start of a job. At each node jobs are executed in input order.

Data transferred by job: Each CPU pulls a job from the queue when it is idle, then it has to wait for an input transfer before the job execution starts.

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

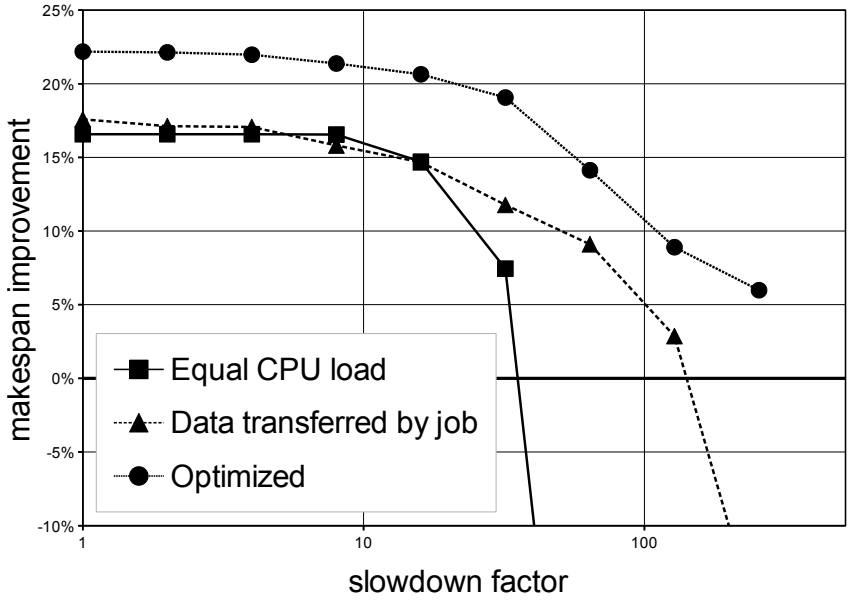

Figure 5: Results of simulations for real data production. Three strategies were evaluated and compared to a ideal local production. The optimized solution (our model) clearly provides the highest gain.

The plot at Figure 5 shows the gain in a makespan delivered by different scheduling policies compared to the job execution at the local site only. The curves shows the performance of the scheduling policies when an overhead of transfer to the remote site increases proportionally to the slowdown factor. When the transfer overhead becomes significant both heuristics ("Equal CPU load" and "Data transferred by job") fail to provide an efficient usage of the remote resources (the makespan improvement goes below zero). Negative makespan improvement means that, in this case, it would be faster to process all the data locally than to distribute it between several sites relying on the heuristic. The proposed global planning approach (Optimized) systematically provides a smaller makespan and adapts to the increase of transfer overheads better then the other simulated heuristics. It was able to provide a positive gain in makespan by using remote resources even when the transfer overhead is comparable to a job duration.

5 Conclusion and future plans

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 allows planning of WAN, storage and CPU loads using the network flow maximization approach. The proposed model will be used in a distributed data production planner which is being developed. The planner 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.

A model for scheduling of data production over Grid was formulated in form of constraint satisfaction problem and solved using constraint programming. The simulations based on data extracted from log files of batch and data management systems of the STAR experiment has shown that the proposed global planning approach systematically provides a smaller makespan and adapts to the increase of transfer overheads better then the other simulated heuristics. The proposed approach can provide an *optimization* and an *automatic adaptation* to fluctuating resources with no need for manual adjustment of a workflow at each site or tuning of heuristics. The future development of global planning for data processing in Grid is ongoing. In future we plan to test this approach on problems of larger size (more nodes, CPU's and links) and improve the search performance in order to enable online scheduling in real environment.

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

[2] Balewski, J., Lauret, J., Olson, D., Sakrejda, I., Arkhipkin, D., et al.: Offloading peak processing to virtual farm by STAR experiment at RHIC. J. Phys.: Conf. Ser. **368**(012011) (2012)

[3] Beaumont, O., Carter, L., Ferrante, J., Legrand, A., Robert, Y.: Bandwidth-centric allocation of independent tasks on heterogeneous platforms. IEEE International Parallel and Distributed Processing Symposium (2002)

[4] Makatun, D., Lauret, J., Rudová, H., Šumbera, M.: Planning for distributed workflows: constraint-based coscheduling of computational jobs and data placement in distributed environments. J. Phys.: Conf. Ser. (2014). Accepted

[5] 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)

[6] Zerola, M., Lauret, J., Barták, R., Šumbera, M.: One click dataset transfer: toward efficient coupling of distributed storage resources and CPUs. J. Phys.: Conf. Ser. **368**(012022) (2012)