One click dataset transfer: toward efficient coupling of distributed storage resources and CPUs.

Michal Zerola[1], **Jérôme Lauret**[2], **Roman Barták**[3], **Michal Šumbera**[1]

[1] Nuclear Physics Institute, Academy of Sciences of the Czech Republic, Czech Republic
[2] Brookhaven National Laboratory, Upton, USA
[3] Faculty of Mathematics and Physics, Charles University, Czech Republic

E-mail: `michal.zerola@ujf.cas.cz`, `jlauret@bnl.gov`, `roman.bartak@mff.cuni.cz`, `sumbera@ujf.cas.cz`

Abstract. The massive data processing in a multi-collaboration environment with geographically spread diverse facilities will be hardly "fair" to users and hardly using network bandwidth efficiently unless we address and deal with planning and reasoning related to data movement and placement. The needs for coordinated data resource sharing and efficient plans solving the data transfer paradigm in a dynamic way are being more required. We will present the work which purpose is to design and develop an automated planning system acting as a centralized decision making component with emphasis on optimization, coordination and load-balancing.

We will describe the most important optimization characteristic and modeling approach based on "constraints". Constraint-based approach allows for a natural declarative formulation of what must be satisfied, without expressing how. The architecture of the system, communication between components and execution of the plan by underlying data transfer tools will be shown. We will emphasize the separation of the planner from the "executors" and explain how to keep the proper balance between being deliberative and reactive. The extension of the model covering full coupling and reasoning about computing resources will be shown.

The system has been deployed within STAR experiment over several Tier sites and has been used for data movement in the favour of user analyses or production processing. We will present several real use-case scenario and performance of the system with a comparison to the "traditional" - solved by hands methods. The benefits in terms of indispensable shorter data delivery time due to leveraging available network paths and intermediate caches will be revealed. Finally, we will outline several possible enhancements and avenues for future work.

1. Introduction

Distributed computing offers large harvesting potential for computing power and brings multiple benefits as far as it is properly exploited. On the other hand it introduces several pitfalls including concurrent access, synchronization, communications scalability as well as specific challenges such as answering key questions like "how to parallelize a task?" knowing where my data and CPU power are located. In data intensive experiments, like the one from High Energy and Nuclear Physics (HENP) community and the STAR [1] [1] experiment, the problem

[1] Solenoidal Tracker at Relativistic Heavy Ion Collider is an experiment located at the Brookhaven National Laboratory (USA). See http://www.star.bnl.gov for more information.

is even more significant since the task usually involves processing and/or manipulation of large datasets.

For the colossal volumes of data being produced every year to be treatable, the technology and system must be manageable. In global collaboration, the needs for coordinated resource sharing and efficient plans solving the problem in a dynamic way are fundamental.

It is apparent that it will be hardly "fair" to users and hardly using network bandwidth efficiently unless we address and deal with planning and reasoning related to data movement and placement. We are addressing the paradigm of distributing the data focusing on one of the largest running physics experiment in the present. The article exploits and applies the solving techniques for designing and building the automated planning and transferring system; enabling scientists to reach fruits of their research leveraging the potential of their resources.

2. Related works and problem analysis

The needs of large-scale data intensive projects arising out of several fields such as bio-informatics (BIRN, BLAST), astronomy (SDSS) or HENP communities (STAR, ALICE) have been the brainteasers for computer scientists for years. Whilst the cost of storage space rapidly decreases and computational power allows scientists to analyze more and more acquired data, appetite for efficiency in Data Grids becomes even more of a prominent need.

Decoupling of job scheduling from data movement was studied by Ranganathan and Foster in [2]. Authors discussed combinations of replication strategies and scheduling algorithms, but not considering the performance of the network. The authors of [3] proposed and implemented improvements to the Condor, a popular cluster-based distributed computing system. The presented data management architecture is based on exploiting the workflow and utilizing data dependencies between jobs through study of related DAGs. Since the workflow in high-energy data analysis is typically simple and embarrassingly parallel without dependencies between jobs these techniques don't lead to a fundamental optimization in this field.

Sato et al. in [4] and authors of [5] tackled the question of replica placement strategies via mathematical constraints modeling an optimization problem in Grid environment. Solving approach in [4] is based on integer linear programming while [5] uses Lagrangian relaxation method [6]. The limitation of both models is a characterization of data transfers which neglects possible transfer paths and fetching data from a site in parallel via multiple links possibly leading to the better network utilization.

We focus on this missing component considering wide-area network data transfers pursuing more efficient data movement. An initial idea of our presented model originates from Simonis [7] and the proposed constraints for traffic placement problem were expanded primarily on links throughputs and consequently on follow-up transfer allocations in time. One of the immense advantages of the constrained based approach is a gentle augmentation of the model with additional real-life rules. Constraints identify the impossible and reduce the realm of possibilities to effectively focus on the possible, allowing for a natural declarative formulation of what must be satisfied, without expressing how.

Designing and implementing the automated planning system in a dynamic environment like Data Grid is, will not be fruitful unless we address the three main issues: *(1) Accuracy of estimation.* Estimating the computation cost of a job (w.r.t. either computing or transfer) is the key success factor, but at the same time the system can be hardly effective if it has to reason about all peculiarities from the environment. The proper abstraction of the real world is needed to provide balance between accuracy and complexity. *(2) Adaptation to dynamic environment.* Pure static approaches assume that resource and task set is given and fixed over time. Since this assumption is not always valid, the adaptation to the changing condition is needed. In other words, the system has to provide proper balance between being **deliberative** and **reactive** at the same time. *(3) Separation of planner from executor.* Fundamentally the first two issues are related to the lack of collaboration between planner and executor. Without a cooperation the planner cannot be aware of the grid environment change and cannot adapt to the more accurate estimations.

3. Architecture

It is important to pay close attention to the architecture of the system - the conceptual glue that holds every phase of a project together. In this section, we will describe the elements of the system, properties and relations between them. We introduce briefly each component following the work-flow (see Fig. 1 for illustration).

Let us start with explaining how requests are put into the system. End users (or stand-alone services) generate requests using the web interface, written in *PHP* following the *MVC* design pattern. There are two possible way how a request can be specified. **a)** either as an encapsulation of the meta-data query (as understood by STAR's File and Replica Catalogue), or **b)** providing the list of files using *filelist.* An example of the catalogue query is:

- **production**=P10ik,
- **filetype**=daq_reco_MuDst,
- **trgsetupname**=AuAu39_production

where we specified type of the production (data set), what type of files we are interested in and some trigger setup. This meta-data query covers about $220,000$ files with a total size of 48TB. The population of the database with files belonging to the request is done asynchronously by separate component as we will see soon.

The second approach of entering the request is using a filelist, which has the following syntax:

```
SITE;STORAGE;PFN
```

Each line then describes exact location of the file given by its physical file name, the storage that holds it and finally the site where the storage is located.

The part of a request is also a desired destination for a data set (in the form of site + storage) which user selects using the web interface.

Afterwards, the request is stored in a *SQL* database (system supports *MySQL* and *PostgreSQL*) in a Catalog agnostic manner (any Catalog should work as far as they have a LFN/PFN concept our approach relies on) with the additional information like user name, group or date of the request.

Later, the component called *File Feeder* contacts the *File and Replica Catalogue* and makes the query for the requested meta-data. The output information is stored back to the database, including all possible locations for every file in a request. This is when population of the internal database with file repositories happens. Because of usually large volume of records that needs to be stored in a database, the *File Feeder* uses `LOAD DATA INFILE` syntax that provides high performance.

The main logic and reasoning about the plan happens in the brain of the system, a component called the *Planner.* It is the place where realization of the model is done and where the plan in iterations is computed. *Planner* takes a subset of all requests for files to be transferred according to the preferred fair-share function. It creates the plan (transfer paths, as we explained in the previous chapter) for the selected requests and stores the plan back to the database.

The individual file transfers are handled by the separate distributed component called *Data Mover.* As we specified at the beginning, we want to use existing point-to-point data transfer tools and use them as the back-end instruments. The *Data Mover* should serve as an intelligent wrapper on top of such tools handling the work. The role of these workers is to perform a point-to-point data transfer on a particular link following the computed plan. The results and

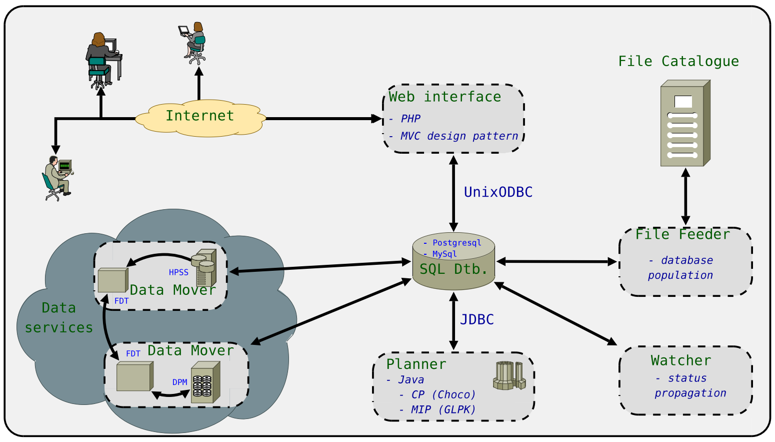

Figure 1. Architecture of the system.

intermediate status is continuously recorded in the database and user can check the progress at any time.

Because of the asynchronous nature of the communication between components, the system has to have well defined states and transitions from one state to another given by state diagrams. We will explain the flow in the following section. The *Watcher*, independent component running at each site, is responsible for changing states of the objects and cache management.

We can see that the whole mechanism is a combination of **deliberative** (assuring optimality) and **reactive** planning (assuring adaptability to the changing environment). Since this is crucial to the argument, in the next section we will describe the respective two components (*Planner* and *Data Mover*) serving up as a “reasoner” and a “worker”.

4. Implementation of the planner

Linear programming is a method of minimizing a given linear function ($\min c^T x$) with respect to the system of linear inequalities ($Ax \leq c$). Vector x represents the variables to be determined. If all can be rational, the problem can be solved in polynomial time. However when some or all of the variables must be integer, corresponding to pure integer or Mixed Integer Programming (MIP) respectively, the problem becomes NP-complete (formally intractable). Several algorithms from operation research are widely used for solving integer programming instances in reasonable time. Reformulation of the problem into the set of linear inequalities often involves relaxation of several constraints. In the following text we introduce the formulation of the data transfer problem into the MIP syntax with involved approximations.

Since required datasets usually overlap together, we would like to minimize also the data movement of the common parts. In other words, if the same file is required by different users and the transfer paths share a link, we transfer the file on common link only once.

For this extension we have to slightly modify the constraint model, since the transfer path for a file can form a *forest* - using the terminology from the graph theory.

We denote the weight of an edge corresponding to the link bandwidth as $\mathbf{bw}(e)$ - bandwidth between two sites or average latency time for the storage elements (e.g. the time to stage the file from the tape system). The information about file’s origins is a mapping of that file to a set of nodes where the file is available.

The input received from the users is a set of file names $\mathbf{F}$, where for every file $f \in \mathbf{F}$ we have a set of sources $\mathbf{orig}(f)$ - sites where the file f is already available and a set of destinations $\mathbf{dest}(f)$ - sites where the file f is supposed to be transferred.

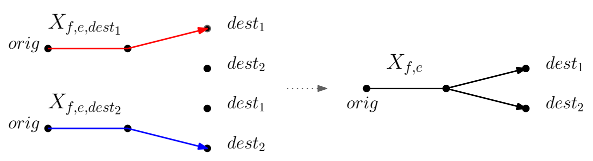

Figure 2. Two independent paths are *glued* together, so the file using their common links will be transferred only once (e.g. the file is staged only once, then transferred to two different destinations)

The essential idea is to use one decision variable for each file, its destination and edge in a graph. We will refer to this $\{0,1\}$ variable as X_{fed}, denoting whether file f is routed (value 1) over the edge e of the network or not (value 0) to its destination d. Mathematical constraints (1-3), ensuring that if all decision variables have assigned values the resulting configuration contains the independent transfer paths, are analogous to the Kirchhoff's circuit laws.

$$\forall f \in \mathbf{F},\ \forall d \in \mathbf{dest}(f): \quad \sum_{e \in \cup \mathbf{OUT}(n | n \in \mathbf{orig}(f))} X_{fed} = 1, \quad \sum_{e \in \cup \mathbf{IN}(n | n \in \mathbf{orig}(f))} X_{fed} = 0 \tag{1}$$

$$\forall f \in \mathbf{F},\ \forall d \in \mathbf{dest}(f): \sum_{e \in \mathbf{OUT}(d)} X_{fed} = 0, \quad \sum_{e \in \mathbf{IN}(d)} X_{fed} = 1 \tag{2}$$

$$\forall f \in \mathbf{F},\ \forall d \in \mathbf{dest}(f),\ \forall n \notin \mathbf{orig}(f) \cup \{d\}: \quad \begin{array}{rl} \sum_{e \in \mathbf{OUT}(n)} X_{fed} & \leq 1 \\ \sum_{e \in \mathbf{IN}(n)} X_{fed} & \leq 1 \end{array} \quad \sum_{e \in \mathbf{OUT}(n)} X_{fed} = \sum_{e \in \mathbf{IN}(n)} X_{fed} \tag{3}$$

Having generated all independent paths for a file to each of its destination, we need to *glue* them together. One can look at it as creating a *forest* using the terminology from the graph theory (Figure 2). We achieve it by defining new binary *two-index* variable X_{fe} stating whether file f uses link e (apart from reasoning about destinations).

$$\forall f \in \mathbf{F},\ \forall e \in \mathbf{E},\ \forall d \in \mathbf{dest}(f): X_{fed} \leq X_{fe} \tag{4}$$

$$\forall f \in \mathbf{F},\ \forall e \in \mathbf{E}: \sum_{d \in \mathbf{dest}(f)} X_{fed} \geq X_{fe} \tag{5}$$

$$\forall f \in \mathbf{F},\ \forall n \notin \mathbf{orig}(f) \cup \{d\}: \sum_{e \in \mathbf{IN}(n)} X_{fe} \leq 1 \tag{6}$$

Finally, since we are minimizing the *makespan*, the time to transfer all files to the requested destinations, we define the constraints (7) for estimation of the completion time **T** variable and appropriate objective function: *minimize* T.

$$\forall e \in \mathbf{E}: \sum_{f \in \mathbf{F}} \frac{\mathbf{size}(f) \cdot X_{fe}}{\mathbf{bw}(e)} \leq T \tag{7}$$

4.1. Performance comparison

In this section we will elaborate on the performance of the automated system under different environment configurations. The principal benefits of the solver depend on leveraging available data services and network links between sites. Let us focus first on performance comparison experienced by relying on data sources with diverse characteristic. All the following measurements were taken in real production environment while monitoring other exterior activities and requests for shared system resources. The monitoring included the control of *HPSS* usage over submitting large-size requests, extensive *WAN* third-party transfers between laboratories, etc. Therefore, the measurements spanning over several hours (and often repeated multiple times) are statistically stable and sound.

The performance of the system can be nicely described by comparing the makespan - the time it takes to bring requested files to the destination. It is important to see also the convergence, how fast were files appearing at the destination. Hence, we will display this tendency as a function of time in the following graphs. Figure 3 is displaying the comparison when the system alternates reasoning about data sources. The transfer was required between STAR *Tier-0* BNL laboratory and *Tier-1* center at LBNL, without involving any third site. We could concentrate thereby entirely on data sources and eliminate the influence of diverse network paths. We were comparing two data services which served as a source of the data. The services are in contrast by different characteristic:

- **HPSS** - holds all the files, works asynchronously. Usually involves waiting time at the beginning upon submission, then high throughput.
- **Xrootd** - holds only the portion of data, works synchronously. Usually provides low latency and high throughput.

The comparison consists of three several hours long transfers when solver was acting always in different mode. The blue line represents the mode when solver was using exclusively only *Xrootd* as the source service. We can see that files started to appear almost immediately at the destination and the slope of the line shows the fastest throughput. For better resolution the small plot in the same figure is displaying the zoomed region in the first 40 minutes. However, since *Xrootd* repository holds only a portion of all data, full data set could not be transferred. What portion of data the service holds usually depend on the age of files. Files from recent datasets are more likely available at *Xrootd* service. The red line represents the mode when system was relying only on HPSS. We can see that there was an initial small waiting time until files started to appear. More important is to realize also the "step-like" trend of the line. This is caused by the HPSS utilization. HPSS prioritizes requests from different users depending on tape locations, history, and other factors; and when there is our requests in turn, it serves the files usually fast. During several hours our request can be postponed and others are prioritized and we have to wait. Unfortunatelly, this "step" can often take several hours, depending on the current circumstances and load. The third black line is representing the last mode, when the system relied on both services concurrently. We can clearly see that the combination of both sources outperforms counting on the stable but not the fastest one (*HPSS*) even if *Xrootd* cannot provide all the files. Realizing which files where reside and coordinating the access to providing services is not something what users can efficiently do by themselves and hence, they often rely on the single one - stable but slow one. This is when automated solver can bring a significant benefit and increase the effectivity of their work.

Let us bring our attention now to the system's reasoning and utilizing diverse network paths. For the purpose of keeping the environment transparent and eliminating the effect of multiple data services, in this case we will concentrate on the sole *Xrootd* service as the source of all files. The slow latency and constant modest bandwidth of this data service allow us to concentrate on the influence of reasoning about network paths. The data transfer scheme is illustrated in Fig.4,

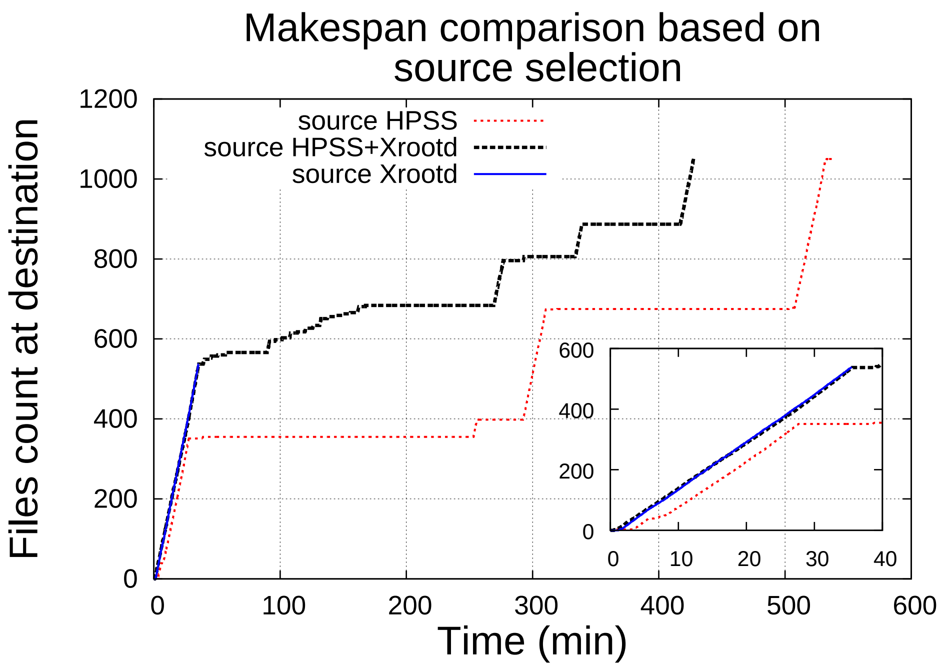

Figure 3. Graph displaying the trend how fast is the plan fulfilled concentrating on 3 different modes of source access. First one uses solely *Xrootd* service, second only the *HPSS* system, and finally third one uses combination of both.

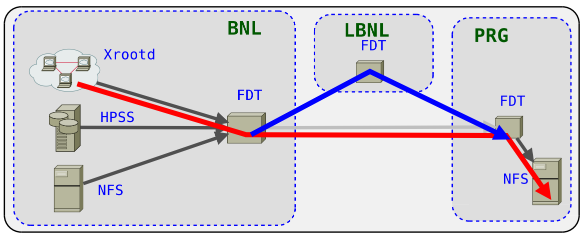

Figure 4. Transfer scheme for diverse network paths. Data movement relies on the *Xrootd* service as the sole source of files while leveraging also intermediate LBNL site.

where data are being moved from BNL laboratory (Tier-0 center) to Prague site (Tier-2 center). The system is allowed to reason also about intermediate site (LBNL laboratory, Tier-1 center) in order to increase the throughput. It is important to state that the connection between the BNL and Prague site is using static routing over dedicated link and is diverse from the path between BNL and LBNL as well as LBNL and Prague (using ESnet, Geant, and CESNET routing). We will again compare the speed how files appear at the destination service while looking at the impact of using intermediate site in parallel.

The graph in Fig.5 is exposing this comparison. The green line represents the mode where only direct BNL to Prague network path was used; while the red line the mode where also additional path through LBNL site was allowed. We can see the clear and very visible benefit in leveraging additional network path and routing part of the traffic via intermediate site. The overall gain in makespan (how less user waited for files) was almost one third of total time comparing to the direct and usual approach.

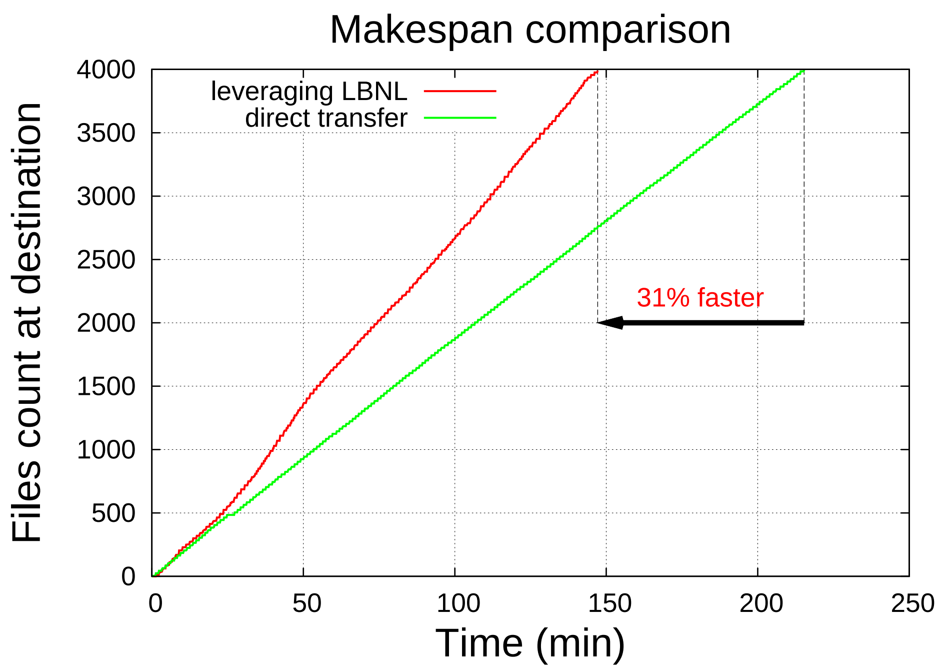

Figure 5. Graph displaying the trend how fast is the plan fulfilled comparing 2 different modes in network paths. Green line represents the mode when only direct transfer to Prague was allowed and red one the mode when part of the traffic was allowed to be routed via LBNL site.

5. Coupling with CPUs

In the previous sections we addressed and focused on the data transfer problem, where the task was to bring data sets to user specified locations. The role of the planner was to decide how to achieve it considering all constraints and having minimal makespan as an objective. However, very often the task is not finished by the time data are moved, but when data are analyzed. In other terms, the data movement itself only precedes the data processing.

Let us underline the benefit of CPU coupling by explaining the real case with production processing in STAR (see Fig. 6). Part of the production is being done at Argonne computing cloud (Chicago) together with PDSF/NERSC computing center (Berkeley). The workflow is following: files are continuously staged from BNL's HPSS system to the local 2TB cache. Since Argonne cloud is not equipped with any cache, the file can be transferred from BNL to Argonne only if there is a free CPU slot. To the contrary, PDSF site has sufficient 20TB cache and can hold data even if all the CPU slots are occupied. Therefore, and **this is being solved by hand**, it turns out that it is advantageous to feed Argonne's CPUs (when free) simultaneously from BNL and PDSF cache. If we had a system capable of dynamically solve this in automatic fashion the benefits could be clearly seen.

While in the pure file transfer problem the task was to locate and bring files to requested destinations, with CPU coupling the problem has to be reformulated. A user doesn't specify a single destination anymore, but a list of available processing sites along the full set of files. Each request R_i is therefore composed of set of files which need to be processed (F_{R_i}) together with a set of destinations - processing sites (D_{R_i}) where user is allowed to run jobs (Eq. 8). By

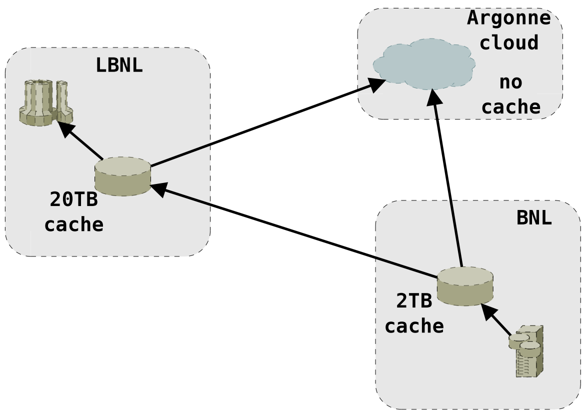

Figure 6. Representation of production processing using Argonne cloud with no cache space and PDSF farm with 20TB cache. Because of limited cache at BNL it turns out practical to feed Argonne cloud with data streaming from both coasts (PDSF and BNL).

saying allowed we mean that user has access and can run jobs on any of these sites.

$$\mathbf{R}_i = \{\underbrace{\{f_1, \ldots, f_{N_i}\}}_{\mathbf{F}_{R_i}}, \underbrace{\{d_1, \ldots, d_{M_i}\}}_{\mathbf{D}_{R_i}}\} \tag{8}$$

The task of the planner is to find transfer paths for all files (every file has to appear in one of the destinations for each request it belongs to) considering the processing phase of the file at the computing site. The system may distribute files from a request among available destinations and execute job on the portion of data set at computing site A while on the other fraction of data set at computing site B.

The MIP model for sloving the pure file transfer problem can be reformulated and extended in such a way that it includes also the CPU reasoning. We will omit here the full set of mathematical constraints due to the length of the article. However, there is one difference in model's logic we would like to point out and explain.

If we look back into our initial motivation from Fig. 6, displaying the production processing schema in STAR, we can see that there is a substantial benefit of bringing files to the storage cache - closer to the processing sites, even if all the slots are used. The model, as it is defined above, reasons about bringing files to the processing sites since they are assigned as the only destinations. If the processing sites are occupied we would still like the solver to reason about bringing files to the storage space closer to the CPUs. In order to modify the reasoning we can create additional *dummy* links from the storages to the *dummy* destination vertex in the graph as represented in Fig. 7. The weights for these additional *dummy* edges need to be properly set so the solver will prioritize bringing files to the cache space in case the processing slots are taken. By setting the appropriate weight we can also control the preference between storage spaces.

6. Conclusions

The work presented in this paper deals and attacks the complex problem of efficient data movements on the network within a distributed environment. The problem itself arises from the real-life needs of the running nuclear physics experiment STAR and it's requirements for data storage and computational power.

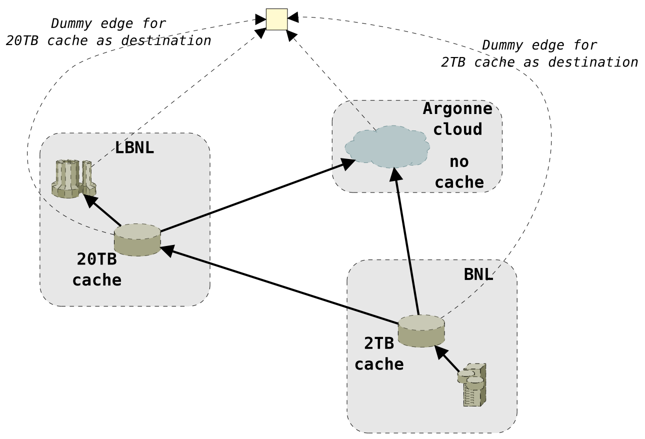

Figure 7. Representation of production processing using Argonne cloud as the graph input for the model. The storage cache nodes are also connected to the *dummy* destination in order to allow the model to bring files closer to the CPUs while they are taken by other jobs.

To our best knowledge, it has been the first time in nuclear physics large scale experiments when automated planning approach was used for reasoning about data transfers and CPU allocations. We presented the underlying model using Mixed Integer Programming techniques and provided the inside view into the design and concept of the architecture. Implementation and coupling of components with standard STAR services proceeded with a real-case evaluation of the framework. We have demonstrated that this concept leads not only to the better comfort for physicists when dealing with data transfers but also brings fruits in the efficiency, load balancing and fail-over handling as was primarily requested.

With upcoming requirements for more frequent Cloud computing where data storage is constrained and the needs for prompt feeding CPUs by data is important, the automated data transfers and job allocation can greatly simplify the user's task. We have addressed this and proposed the extension of the model for reasoning about computational power as well.

The journey to the fully automated and intelligent system scheduling user's tasks considering all relevant constraints is certainly long and there are still numerous aspects that need to be addressed. We believe that the presented work contributed to this collaborated effort with several sound ideas, techniques and concepts.

Acknowledgments
The work has been supported in part by the Offices of NP and HEP within the U.S. DOE Office of Science and the U.S. NSF and by the grants LC07048 and LA09013 of the Ministry of Education of the Czech Republic.

References

[1] Adams S C J 2005 *Nuclear Physics A* **757** 102 URL `doi:10.1016/j.nuclphysa.2005.03.085`
[2] Ranganathan K and Foster I 2002 *11th IEEE Symposium on High-Performance Distributed Computing* vol 0 (Los Alamitos, CA, USA: IEEE Computer Society) pp 352–358 ISSN 1082-8907
[3] Shankar S and DeWitt D J 2007 *HPDC '07: Proceedings of the 16th international symposium on High performance distributed computing* (New York, NY, USA: ACM) pp 127–136 ISBN 978-1-59593-673-8
[4] Sato H, Matsuoka S, Endo T and Maruyama N 2008 *GRID* (IEEE) pp 250–257
[5] Rahman R M, Barker K and Alhajj R 2007 *CCGRID* (IEEE Computer Society) pp 171–178
[6] Fisher M L 1981 *Management Science* **27** 1–18
[7] Simonis H 2006 *Handbook of Constraint Programming* ed Rossi F, van Beek P and Walsh T (Elsevier) chap 25, pp 875–903