Planning for distributed workflows: constraint-based coscheduling of computational jobs and data placement in distributed environments

Dzmitry Makatun[1,3], **Jérôme Lauret**[2], **Hana Rudová**[4] **and Michal Šumbera**[3]

[1]Faculty of Nuclear Physics and Physical Engineering, Czech Technical University in Prague
[2]STAR, Brookhaven National Laboratory, USA
[3]Nuclear Physics Institute, Academy of Sciences, Czech Republic
[4]Masaryk University, Czech Republic

E-mail: `dzmitry.makatun@fjfi.cvut.cz`

Abstract. When running data intensive applications on distributed computational resources long I/O overheads may be observed as access to remotely stored data is performed. Latencies and bandwidth can become the major limiting factor for the overall computation performance and can reduce the CPU/WallTime ratio to excessive IO wait. Reusing the knowledge of our previous research, we propose a constraint programming based planner that schedules computational jobs and data placements (transfers) in a distributed environment in order to optimize resource utilization and reduce the overall processing completion time. The optimization is achieved by ensuring that none of the resources (network links, data storages and CPUs) are oversaturated at any moment of time and either (a) that the data is pre-placed at the site where the job runs or (b) that the jobs are scheduled where the data is already present. Such an approach eliminates the idle CPU cycles occurring when the job is waiting for the I/O from a remote site and would have wide application in the community. Our planner was evaluated and simulated based on data extracted from log files of batch and data management systems of the STAR experiment. The results of evaluation and estimation of performance improvements are discussed in this paper.

1. Introduction

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)[1, 2]. 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.

Long I/O overheads when accessing data from remote site can significantly reduce the application's CPUtime/WallTime ratio [3, 4]. 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 [5]. 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.

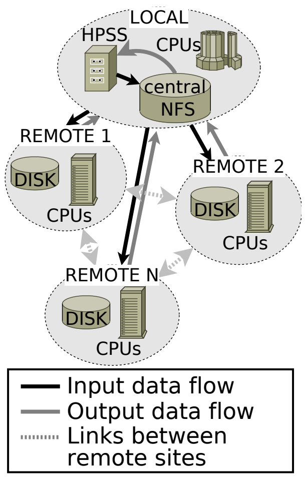

Figure 1. Schema of data production in the Cloud.

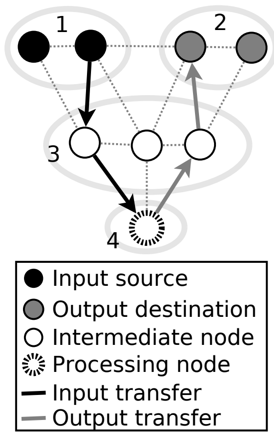

Figure 2. An example of a transfer path. Illustration for constraints 1-4 in section 2.

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?

Problems of scheduling, planning and optimization are being commonly solved with the help of Constraint Programming (CP) [6]. 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.

2. Model formulation, assumptions and search approach

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 [1] 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.

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:

(i) 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.

(ii) 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.

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.

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}	$Fdur_{fc}$	$size_f$	$disk_c$

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 [7].

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.

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).

3. Simulations

The constraint satisfaction problem was implemented using MiniZinc [8] and Gecode [9] 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) [10]. 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.

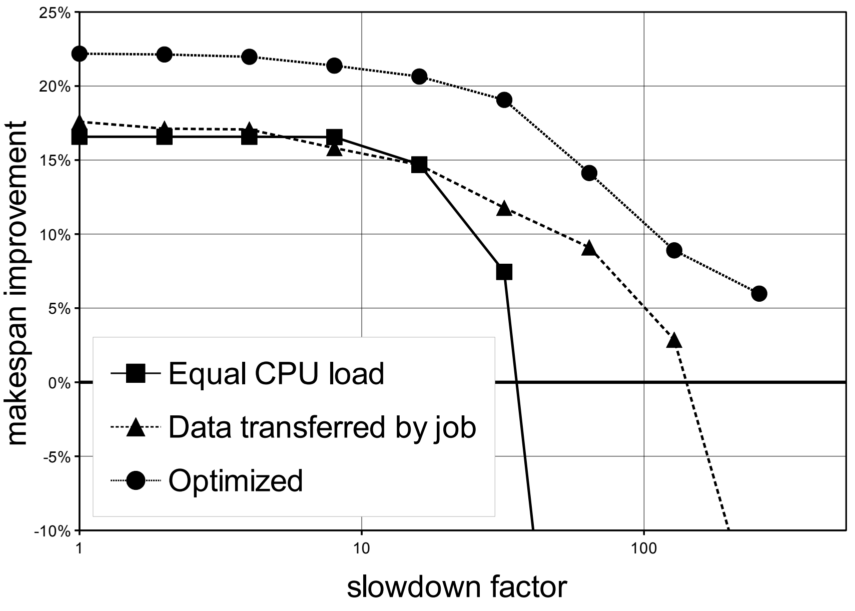

Figure 3. Results of simulations for real data production. Three strategies were evaluated and compared to a ideal local production. The optimized solution (our model) clearly provides the highest gain.

The plot at Figure 3 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.

4. Conclusion

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.

Acknowledgments

This work has been supported by the Czech Science Foundation (GA-CR) and the Office of Nuclear Physics within the U.S. Department of Energys. We would like to acknowledge and thank Brookhaven National Laboratory for resources that they made available.

References

[1] Zerola M, Lauret J, Barták R and Šumbera M 2012 One click dataset transfer: toward efficient coupling of distributed storage resources and CPUs *J. Phys.: Conf. Series* **368** 012022
[2] Makatun D, Lauret J and Šumbera M 2014 Study of cache performance in distributed environment for data processing, *J. Phys.: Conf. Series* **523** 012016
[3] Horký J, Lokajíček M and Peisar J 2013 *Influence of Distributing a Tier-2 Data Storage on Physics Analysis* (Beijing: 15th Int. Workshop on Advanced Computing and Analysis Techniques in Phys. Res.)
[4] Betev L, Gheata A, Gheata M, Grigoras C and Hristov P 2014 Performance optimisations for distributed analysis in ALICE *J. Phys.: Conf. Series* **523** 012014
[5] Balewski J, Lauret J, Olson D, Sakrejda I, Arkhipkin D, Bresnahan J, Keahey K, Porter J *et al.* 2012 Offloading peak processing to virtual farm by STAR experiment at RHIC *J. Phys.: Conf. Series* **368** 012011
[6] Rossi F, Beek P and Walsh T 2006 *Handbook of Constraint Programming* (Amsterdam: Elsevier)
[7] Troubil P and Rudová H 2011 Integer Linear Programming Models for Media Streams Planning. (Int. Conf. on Applied Operational Res.) *Lecture Notes in Management Sc.* **3** 509-22
[8] Nethercote N, Stuckey P, Becket R, Brand S, Duck G and Tack G 2007 MiniZinc: Towards a Standard CP Modelling Language *Lecture Notes in Comp. Sc.* **4741** 529-543
[9] Tack G 2008 *Gecode: an Open Constraint Solving Library* (Paris: OSSICP08 Workshop at CP-AI-OR08)
[10] Korea Institute of Science and Technology Information (KISTI) http://en.kisti.re.kr/