Using constraint programming to resolve the multi-source / multi-site data movement paradigm on the Grid

Michal ZEROLA[*]
Nuclear Physics Institute, ASCR
E-mail:`michal.zerola@ujf.cas.cz`

Jérôme Lauret
Brookhaven National Laboratory
E-mail: `jlauret@bnl.gov`

Roman Barták
Faculty of Mathematics and Physics, Charles University
E-mail: `bartak@kti.mff.cuni.cz`

Michal Šumbera
Nuclear Physics Institute, ASCR
E-mail: `sumbera@ujf.cas.cz`

In order to achieve both fast and coordinated data transfer to collaborative sites as well as to create a distribution of data over multiple sites, efficient data movement is one of the most essential aspects in distributed environment. With such capabilities at hand, truly distributed task scheduling with minimal latencies would be reachable by internationally distributed collaborations (such as ones in HENP) seeking for scavenging or maximizing on geographically spread computational resources. But it is often not all clear (a) how to move data when available from multiple sources or (b) how to move data to multiple compute resources to achieve an optimal usage of available resources. Constraint programming (CP) is a technique from artificial intelligence and operations research allowing to find solutions in a multi-dimensional space of variables. We present a method of creating a CP model consisting of sites, links and their attributes such as bandwidth for grid network data transfer also considering user tasks as part of the objective function for an optimal solution. We will explore and explain trade-off between schedule generation time and divergence from the optimal solution and show how to improve and render viable the solution's finding time by using search tree time limit, approximations, restrictions such as symmetry breaking or grouping similar tasks together, or generating sequence of optimal schedules by splitting the input problem. Results of data transfer simulation for each case will also include a well known Peer-2-Peer model, and time taken to generate a schedule as well as time needed for a schedule execution will be compared to a CP optimal solution. We will additionally present a possible implementation aimed to bring a distributed datasets (multiple sources) to a given site in a minimal time.

XII Advanced Computing and Analysis Techniques in Physics Research
November 3-7, 2008
Erice, Italy

1. Introduction

1.1 Problem area

Computationally challenging experiments such as the one from the High Energy and Nuclear Physics (HENP) community have developed a distributed computing approach (a.k.a. Grid computing model) to face the massive needs of their Peta-scale experiments. The era of data intensive computing has surely opened a vast arena for computer scientists to resolve practical and exciting problems. One of such HENP experiments is the STAR[1] (Solenoidal Tracker at Relativistic Heavy Ion Collider [1]) experiment located at the Brookhaven National Laboratory (USA).

In addition to a typical Peta-byte scale challenge and large computational needs this experiment as a running experiment acquires a new set of valuable experimental data every year, introducing other dimension of safe data transfer to the problem. From the yearly data sets, the experiment may produce many physics-ready derived data sets which differ in accuracy as the problem is better understood and as time passes. Thus, demands for a large-scaled storage management and efficient scheme to distribute data grows as a function of time, while on the other hand, end-users may need to access data sets from previous years and consequently access data at any point of the data production timeline. Coordination is needed to avoid random access destroying efficiency.

The user's task is typically embarrassingly parallel; that is, a single program can run *N* times on a fraction of the whole data set split into *N* sub-parts with usually no impact on science reliability, accuracy, or reproducibility. For a computer scientist, the issue then becomes how to split the embarrassingly parallel task into *N* jobs in the most efficient manner while knowing the data set is spread over the world and/or how to spread 'a' dataset and place best the data for maximal efficiency and fastest processing of the task.

The purpose of this work is to design and develop an automated system that would efficiently use all available computational and storage resources. It will relieve end users of making decisions among possible ways of their task execution (which includes locating and transferring data to desired sites that appear optimal to user) while preserving fairness. Users' knowledge of the whole system and data transfer tools will be reduced just to the communication with the future planner that will guarantee its decision to spread the task and data sets over chosen sites was, under current circumstances, the most efficient and optimal.

1.2 Milestones

Rather than trying to solve the problem directly from a task scheduling perspective within a grid environment, we split the problem into several stages. By isolating the data transfer/placement and the computational challenges from each other, we get an opportunity to study the behavior of both sets of constraints separately.

Individual tasks depend on a datasets which size has to be considered as well, since the time required for its staging and transfers is also significant. Therefore, the **first milestone** is to design and develop a data transfer planner/scheduler. For a given dataset needed at a given site, its aim is to create a plan with an objective to prepare and move the files from that dataset within the

[*]Speaker.

[1]http://www.star.bnl.gov

shortest time. The next requirement is to define and achieve fair share transfers within a multiuser environment. This means that if one user asked for a huge amount of data to be transferred at a site, then another user who asked just for one file shouldn't wait until the first user's plan is finished.

The **next milestone** generalizes data transfer planning between sites. The goal for this stage is not to transfer files to one particular site, but do the transfer to several destinations. More precisely, the planner's goal is to achieve presence of each file (from user's input task) at one out of all possible destinations, while still having the objective in mind, to minimize the finish time of the last file transfer the user waits for.

The second milestone is highly correlated with the **final milestone -** scheduling the data transfers together with particular tasks (jobs) on a grid. The subtask is not finished after a file is transferred at a destination site, but when the user's job executed at the same site (and dependent on this file) is finished. Thus, the planner still has the freedom of choosing a destination site for each file, but it has to consider that each site has a specific characteristic of its computational performance. These attributes include, for example, the number of available CPUs or the actual load, so it can be more effective to transfer some files over the slower link to the computationally high performance site (or vice versa). The final objective is to minimize the finish time of the last user's job. In this article we focus on the first milestone.

2. Problem formalization

In this section, we will present a formal description of the problem and an approach based on Constraint Programming technique used in artificial intelligence and operations research. Using this technique, we search for assignment of given variables from their domains, in such a way that all constraints are satisfied and value of an objective function is optimal [4].

We will introduce the transfer network consisting of sites holding information which files are available at the site. For each file we will search for a path leading to the destination and time slots for each link on the transfer path, when a particular file transfer should occur.

The network consists of a set of nodes **N** and a set of directed edges **E**. The set **OUT**(n) consists of all edges leaving node n, the set **IN**(n) of all edges leading to node n. Input received from a user is a set of file names needed at a destination site **dest**. We will refer to this set of file names as to demands, represented by **D**. For every demand d we have a set of sources (**orig**(d)), sites where the file (d) is already available. We will use one decision variable for every demand and link of the network (edge in graph). The $\{0,1\}$ variable X_{de} denotes whether demand d is routed over edge e of the network. The second variable $start_{de}$ denotes start time of transfer corresponding to the demand d over edge e. More approaches can be found in [6].

$$\min_{X_{de},start_{de}} \max_{e\in\mathbf{E}} \underbrace{\left(start_{de} + \frac{size(d)}{speed(e)}\right)}_{end_{de}} \cdot X_{de} \tag{2.1}$$

$$\forall d \in \mathbf{D}: \sum_{e\in\cup\mathbf{OUT}(n|n\in\mathbf{orig}(d))} X_{de} = 1, \quad \sum_{e\in\cup\mathbf{IN}(n|n\in\mathbf{orig}(d))} X_{de} = 0 \tag{2.2}$$

$$\forall d \in \mathbf{D}: \sum_{e\in\mathbf{OUT}(dest(d))} X_{de} = 0, \quad \sum_{e\in\mathbf{IN}(dest(d))} X_{de} = 1 \tag{2.3}$$

$$\forall d \in \mathbf{D}, \forall n \notin \{orig(d) \cup dest(d)\} :$$
$$\sum_{e \in \mathbf{OUT}(n)} X_{de} \leq 1, \sum_{e \in \mathbf{IN}(n)} X_{de} \leq 1, \sum_{e \in \mathbf{OUT}(n)} X_{de} = \sum_{e \in \mathbf{IN}(n)} X_{de} \tag{2.4}$$

$$\forall e \in \mathbf{E}, \forall d \in \mathbf{D} : X_{de} = 1 : \bigcap [start_{de}, \underbrace{start_{de} + \frac{size(d)}{speed(e)}}_{end_{de}}] = \emptyset \tag{2.5}$$

$$\forall n \in \mathbf{N}, \forall d \in \mathbf{D} : \sum_{e \in \mathbf{IN}(n)} \underbrace{\left(start_{de} + \frac{size(d)}{speed(e)} \right)}_{end_{de}} \cdot X_{de} \leq \sum_{e \in \mathbf{OUT}(n)} start_{de} \cdot X_{de} \tag{2.6}$$

$$X_{de} \in \{0,1\}$$
$$start_{de} \in \mathcal{N}^{+}$$

The *path constraints* (2.2, 2.3, 2.4) state that there is a single path for each demand (path starting right in one of origin sites, leading to the destination). Equation (2.5) ensures there is only one active file transfer over every edge in time. The last equation states that a transfer of the file at any site can start only if the file is already available at the site (Eq. 2.6)(i.e., a transfer of the file to this site has finished). The objective (Eq. 2.1) is to minimize the latest finish time of transfer over the whole files.

2.1 Constraint model

For implementation of the solver we use Choco [2], a Java based library for constraint satisfaction problems, constraint programming and explanation-based constraint solving. Among 70 available constraints Choco also provides a rich set of constraints for scheduling and resource allocation needed for this project. Closer illustration of several Choco uses can be found in [2], [3], and [7]. In addition, choosing a Java based platform allows for an easier integration of our planner with the tools currently used in the STAR environment.

Constraints introduced in the previous section were used directly via appropriate Choco structures, except the equation 2.5, which ensures at most single file transfer in any time on any link. For this, we used the **cumulative** scheduling constraint and notation of tasks and resources. Tasks are represented by their duration, by ranges for starting and ending times, and by resource consumption respectively. They are allocated to the resource(s) in such a way that at any time resource capacities cannot be exceeded.

In our case, each link acts as a separate resource with capacity 1 (unary resource) and each file demand creates a single task on every resource, which duration depends on the current link speed (resource characteristic) and consumption of the resource corresponds to the value of variable X, i.e. no consumption if the transfer path for demand does not include current link (resource), or consumption 1 otherwise. Figure (1) depicts one possible schedule for transferring a file (F) with an origin at $Site_1$ and $Site_2$ to destination $Dest$. Values of the X variables define the path, while the resource profile for each link is on the right side of the figure.

[2]http://choco.sourceforge.net

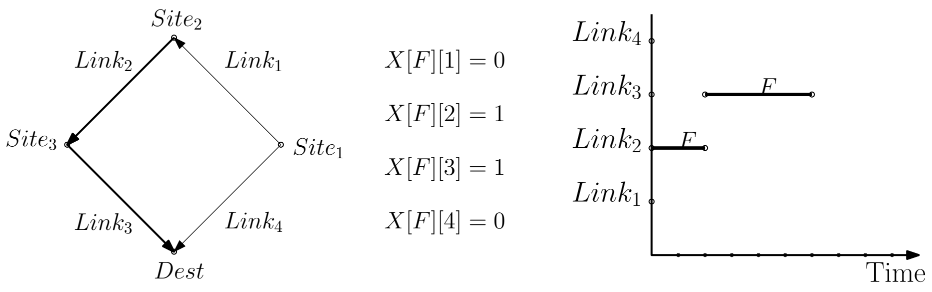

Figure 1: Example of a schedule solution with file F and its origin at $Site_1$ and $Site_2$. Variables X represent selected path transfer, in this case via links $Link_2$ and $Link_3$. On the right side is a corresponding Gantt chart of the schedule.

The search strategy, following Choco notation, is split into two *goals*. First one is to find assignment for X variables, i.e. paths for each transfer, while the second is to allocate time slot, assign *start* variables, for each transfer at chosen links. For both goals the default *'minimum domain'* variable selection and *'increasing value'* value selection heuristic were used.

3. Direct connections

In order to closely analyze the problem and its scale as well as the behavior of the technique we used, several restrictions that simplify the problem were imposed to the model. We started to explore the network, where only **direct connections** for data movement are allowed. In other words, file can not be transferred from its origin to the destination by a path longer than one.

One can think that such a restriction shrinks the search space enormously, but closer look reveals that the number of possible combinations is still large.

Let's suppose that we have a network of 5 sites, all connected to the destination and 100 files available at each site ($|orig(f)| = 5$). The number of decision variables X is therefore 500 ($= |D| * |E|$). Even if an upper bound for all possible combinations (2^{500}) is reduced by a propagation to 5^{100} (solver has a freedom of 5 choices of an origin for each file), brute-force methods can run 'forever'.

With the intent to stay close to reality, we fixed the number of sites to 5, which approximately represents the number of sites currently available in the STAR experiment. For each link we introduced a *slowdown factor* that influences the transfer time needed to move the data over this link. A slowdown factor 1 means that file of size 1 unit can be transferred in 1 unit of time, but with a slowdown factor 4 only in 4 units of time, etc...

Considering the file demands, we studied the following cases: a) every file is available only at one particular site; b) file is available at sites given by a probability function that represents the reality; c) file is available at all sites. We will name these scenarios as **distinct**, **weighted** and **shared** respectively. For all cases we fixed the file size to a 1 unit, i.e. all files have the same size.

3.1 Shared links

So far we have assumed that all links incoming or outgoing from any site have their own bandwidth (slowdown factor) that is not affected by other ones. However, in reality this is not always feasible, since several links leading to a site usually share the same router and/or physical fiber which bandwidth (capacity) is less than the sum of their own values. Hence, simultaneously one cannot use all links at their maximum bandwidths. We express this constraint by adding an additional resource per each group of shared links. Capacity of the resource will be the bandwidth of a shared link or a router, while tasks correspond to the scheduled transfers using any link belonging to this group with consumptions equal to its slowdown factor.

3.2 Reducing the search time

We studied also several techniques for reducing the time spent during a search.

One of the common techniques for reducing the search tree is detecting and breaking variable symmetries. This is usually done by adding variable symmetry breaking constraints that can be expressed easily and propagated efficiently using lexicographic ordering. One idea that can be applied in the studied case (direct connections and fixed file size) is following: if two files have the same origin sets, links selected for the first file and for the second one respectively must be ordered. The reason behind is that both files must be transferred to the destination and their size is equal, it is not necessary to also check the symmetrical swapped case, since the transfer time can not be shorter.

Another approach is based on the idea, where instead of searching for a global optimal solution that can be very computing time consuming, we try to find an optimal solution for smaller parts of the input whereas and due to combinatorial effects, the sum of all times spent to solve a portion of the plan will be just a fraction of the time needed otherwise for the full plan. This principle is even more suitable for our needs, since network link speeds vary in time and some sites can be down after the schedule is produced. Generally, transferring all data files takes a significant amount of time and, during this time, a lot of environmental conditions (site status, network load) can be different to the ones the scheduler considered at the beginning. Thus the computed optimal schedule for the full input is not necessarily the best approach as after a lapse time, it may no longer be valid due to externally rapidly changing conditions. Chunk scheduling also allows for a better fair-shareness as new requests by users may be bundled at a later (human reasonable) time without the need to have all previous requests being honored first (which may happen after a long period of time, hence a slow reaction time of the system allowing at best a FIFO approach).

One of the requirements for being able to split the input files into chunks and producing an optimal schedule is for each chunk to be resolved independently while propagating the results to the next chunk. More precisely, one of the results of the scheduler for a given chunk of files is the information of the computed starting/ending times for each file using particular links. In other words, to be able to consider the historical usage of the links, the current solver receives times at which the links will be busy, thus further scheduling for current chunk cannot place file transfer in these time-slots (information propagation is hence achieve). We implement this by allocating a “fake tasks” that is, hypothetical tasks with fixed starting and ending times that were propagated from previous schedules (Figure 2).

Also limits can be imposed on the search algorithm to avoid spending too much time in the exploration. One of them is fixing the time limit on a search tree. When the execution time is equal to the time limit, the search stops whether an optimal solution is found or not. One of the

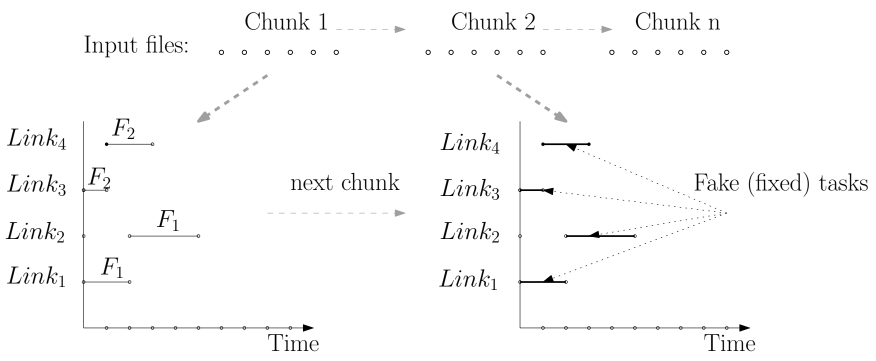

Figure 2: Allocating fake tasks according to the previous schedule. Input files are split into batches and each batch (chunk) is planned separately respecting the results of previous calls.

algorithms we studied was based on this, with a time-limit linearly dependent on the number of files in a request.

4. Directed (simple) paths

Considering the model, no changes are necessary to perform in order to allow solver search for transfer paths longer than one. However, since data set transit takes storage space from the intermediate site, one must be sure that during file transfer from site A to C, using site B, there is enough space at the intermediate site B to hold the file in transit.

4.1 Storage capacity

In order to respect the storage restrictions we introduce the next attribute for each site, the available (free) space, or the storage capacity. All the time during the execution of a schedule, the storage capacity constraint for each site must be respected.

For each site we consider all possible ways (pairs of *inLink* and *outLink* how a file can be transferred trough it). Whether or not a pair is really used for the demand *d* is expressed by *channelingVariable*, which also defines the consumption of the task (Figure: 3).

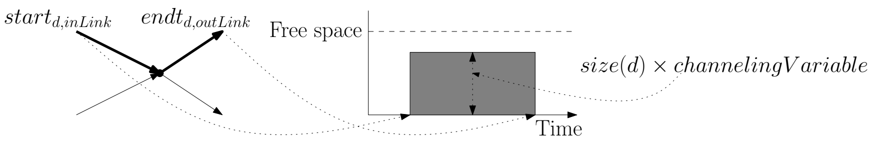

Figure 3: Storage resource and representation of the consumption of a free space at site during transferring a file *d* through it. The resource consumption is defined by start time of the file transfer over *inLink* and finish time of its transfer over *outLink*.

If the pair is not used, the consumption is set to zero and storage resource is invariable to this task, otherwise the consumption is set to the file size.

5. Comparative studies

In this section we present the performance comparison of several methods of the CSP solver introduced in previous sections as well as of the Peer-2-Peer simulator. We will also show an effect of one constraint (storage based) for a simple paths case and an example of the optimal schedule produced by the solver.

5.1 Peer-2-Peer simulator

To provide a base comparison with the results of our CSP based solver we chose to implement a Peer-2-Peer (P2P) model. This model is well known and successfully used in similar fields like file sharing, telecommunication, or media streaming. We implemented a P2P simulator by creating the following work-flow: **a)** put an observer for each link leading to the destination; **b)** if an observer detects the link is free, it picks up the file at his site (link starting node), initiate the transfer, and waits until the transfer is done. We introduced a heuristic for picking up a file as typically done for P2P. Link observer picks up a file with a smallest cardinality in the sense of its $|origin|$, i.e. the file that is available at the smallest number of sites and if there are more files available with the same cardinality, it randomly picks any of them. After each transfer, the file record is removed from the list of possibilities over all sites. This process is typically resolved using distributed hash table (DHT) [5], however in our simulator only simple structures were used. Finally an algorithm terminates when all files reach the destination, thus no observer has any more work to do.

5.2 Results

In Figure 4, we show a comparison of times needed to produce the schedules and divergence of the results (makespan) to the optimal solution between several algorithms. We present the results only for the **weighted** case with direct connections and will only describe the qualitative features for the other cases. Weights (probabilities) that were used for sites were $1.0, 0.6, 0.01$, and 0.01.

The X axes denote the number of files in a request while Y is the time (in units) needed to generate the schedule and percentage loss on optimal solution. We can see that time to find an optimal schedule without any additions grows exponentially and is usable only for a limited number of files, 50 in the weighted case and 20 in the shared case. This difference is induced by a higher number of possible configurations as long as any site can be selected as an origin. By introducing symmetry breaking, the solving time is improved, but still not usable for more than 200 files. Using a time-limit on the other hand we are moving apart from an optimal solution with increasing files in request, which is even more visible in the shared case. Thus setting the time-limit as a linear function to the number of files, while using a default search strategy based on minimal domains, is not sufficient.

In contrast, splitting the input into chunks is giving the best performance results both in the running time and also in the quality of the makespan. Even scheduling by chunk of size 1, i.e. file by file, doesn't produce worse result than using larger chunks – this is explained and mainly due to the use of the propagation of information from previous steps and link conditions (and statistically large samples, rendering localized differences impossible to differentiate). While we note as well the efficacious performance of a simple P2P algorithm, it is worth to mention that this model is usable only in a direct connection case, while our intent is to study more complex networks with much more restrictions and hence, this approach cannot be retained to first order.

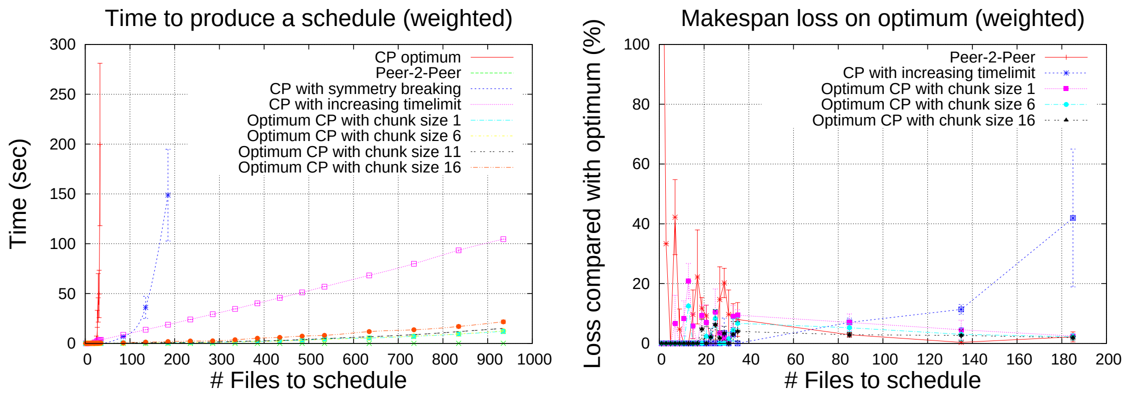

Figure 4: Approximation of the solver's runtime depending on different strategies (left) and corresponding loss of the makespan comparing to an optimal schedule (right) for the weighted case.

To see the real effect of the storage constraint, in Gantt charts (Figure 5) are shown two schedules (without and with enabled constraint) for the same dataset, considering the funnel network displayed in the upper part of the figure with a limited available space at $Site_3$ only for one file size unit. This extreme example permits only a single transfer via site $Site_3$, that fills available space until a file is fully transfered to the destination $Site_4$. After that, the space at $Site_3$ is again released and another file can go trough.

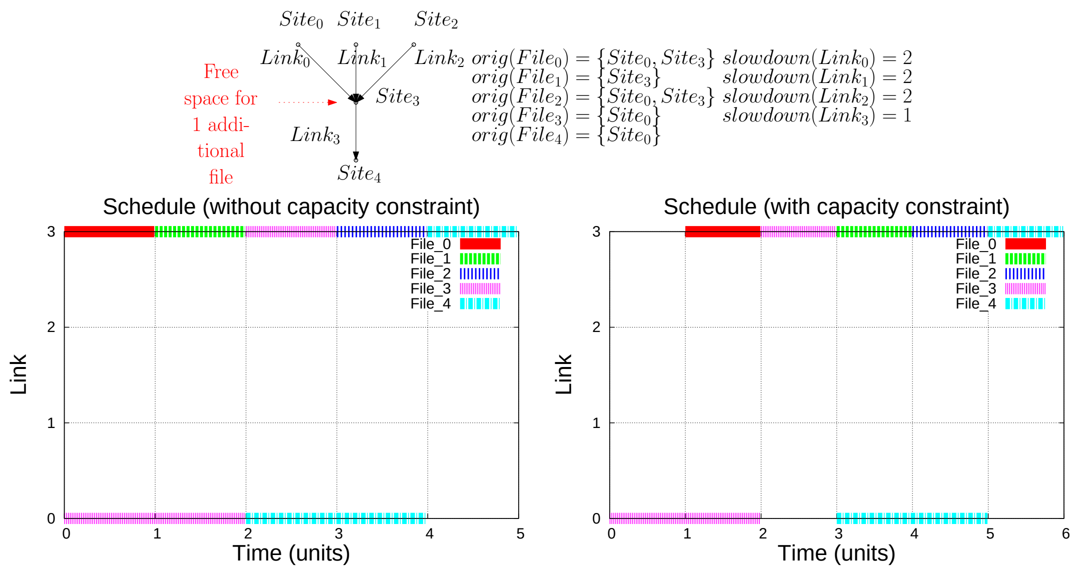

Figure 5: Gantt chart of a schedule without storage constraints (left) and a schedule with storage constraints (right) generated on the funnel network with limited storage capacity (up). We can see that with a constraint, the solver is respecting the limited space at $Site_3$, therefore the start of a transfer of $File_4$ is postponed till $File_3$ has reached the destination.

6. Conclusion

We presented an approach using a Constraint Programming model to tackle the problem of efficient data transfers/placements and job allocations problem within a distributed computing environment. Usage of constraints and declarative type of programming offers straightforward ways of representing many real life restrictions which is also less vulnerable to software coding errors in an ever expanding framework. On the other hand, since a search space is usually extensive, methods like symmetry breaking or approximations and understanding the scale of the problem are fundamental. In this work, we showed that using the scheduling of data transfers by sequence of smaller chunks gives results close to the optimal solution and provides very acceptable running time performance. We have further implemented several constraints for dealing with shared network links or limited storage capacities at sites and actual results are promising. More work will be needed to demonstrate the full power and usability of constraint programming especially in reducing search times and using smart heuristics but insofar, our results are promising. Not only such real-life constraints can be easily modeled and the formalism straight forward but the results and implementation showed a workable proof of principle of this approach.

7. Acknowledgements

The investigations have been partially supported by the IRP AVOZ10480505, by the Grant Agency of the Czech Republic under Contract No. 202/07/0079, by the grant LC07048 of the Ministry of Education of the Czech Republic, by the HENP Divisions of the Office of Science of the U.S. DOE and by the U.S. NSF.

References

[1] STAR Collaboration: J. Adams. Experimental and theoretical challenges in the search for the quark gluon plasma: The STAR collaboration's critical assessment of the evidence from RHIC collisions. *Nuclear Physics A*, 757:102, 2005.

[2] David Benavides, Sergio Segura, Pablo Trinidad Martín-Arroyo, and Antonio Ruiz Cortés. Using Java CSP solvers in the automated analyses of feature models. In Ralf Lämmel, João Saraiva, and Joost Visser, editors, *GTTSE*, volume 4143 of *Lecture Notes in Computer Science*, pages 399–408. Springer, 2006.

[3] Alexander Lazovik, Marco Aiello, and Rosella Gennari. Choreographies: using constraints to satisfy service requests. In *AICT/ICIW*, page 150. IEEE Computer Society, 2006.

[4] K. Marriott and P. Stuckey. *Programming with Constraints*. MIT Press, Cambridge, Massachusetts, 1998.

[5] Naor and Wieder. A simple fault tolerant distributed hash table. In *International Workshop on Peer-to-Peer Systems (IPTPS), LNCS*, volume 2, 2003.

[6] Helmut Simonis. Challenges for constraint programming in networking. In Mark Wallace, editor, *CP*, volume 3258 of *Lecture Notes in Computer Science*, pages 13–16. Springer, 2004.

[7] Jules White, Douglas C. Schmidt, Krzysztof Czarnecki, Christoph Wienands, Gunther Lenz, Egon Wuchner, and Ludger Fiege. Automated model-based configuration of enterprise Java applications. In *EDOC*, pages 301–312. IEEE Computer Society, 2007.