Efficient Multi-Site Data Movement Using Constraint Programming for Data Hungry Science

Michal Zerola[1], Jérôme Lauret[2], Roman Barták[3] and Michal Šumbera[1] for the STAR Collaboration

Nuclear Physics Institute [1]
Academy of Sciences of the Czech Republic

Brookhaven National Laboratory [2]
USA

Faculty of Mathematics and Physics [3]
Charles University, Czech Republic

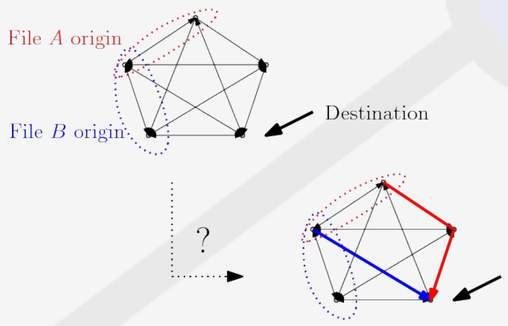
Motivation

Computationally challenging experiments such as the *STAR* at *RHIC* (Relativistic Heavy Ion Collider located at the Brookhaven National Laboratory (USA)), have developed a distributed computing approach (*Grid*) to face their massive computational and storage requirements. Since the peta-bytes of data produced by *STAR* are geographically spread, it is necessary to face the question of efficient data transfers and placements in order to bring requested dataset to a particular site for further analyses.

• How to transfer a set of files for a physicist to his site in a **shortest time** possible for fast analysis turn around?

• How to achieve **controlled planning** of data-movement between sites on a grid (not necessarily following Tier architecture)?

• Ultimately: How to **couple storage** and **computing element** aspects for distributing data on the grid for further processing?



We concentrate on the first (most practical and of immediate need) task:
· create a plan how to transfer data from data-warehouses to the requested site in the shortest time ·

Realization

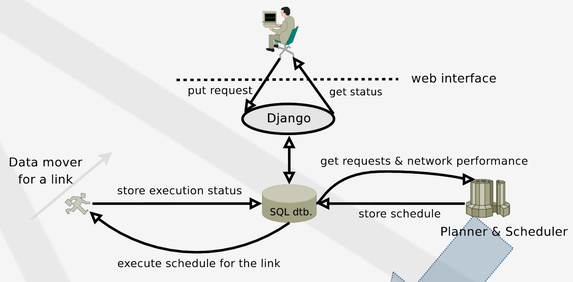
Scope of the architecture

• user has intent to work on a dataset either for analysis or data production - request is made, dataset information is stored in a database

• the intelligence is embedded inside the Planner & Scheduler component

• for an actual *"transfer bulk"* the schedule is computed and stored into the database

• data-mover daemon takes the schedule related to its particular link and executes the transfers (transfer is delegated to the mover)



We concentrate on the **brain** component responsible for generating the schedule for requested files.

Formal model

The input for our problem consists of two parts:
• the first component of the information needed relates to the network - formally modeled as a **directed weighted graph**, it consists of a set of nodes **N** (sites) and a set of directed edges **E** (links). The weight of an edge describes the number of time units needed to transfer one size unit.
• the second ingreditent is information about the dataset and the files to be transferred: we need to know where the files reside (mapping)

Two iterative stages:
• a transfer path for each file, i.e., one origin and a valid path from the origin to the destination, is selected (**planning**)
• for each file and its selected transfer path, the particular transfers via links are scheduled in time such that the resulting plan has minimal makespan (**scheduling**).

Algorithm 1 Pseudocode for a search procedure.

```
makespan ← sup
plan ← Planner.getFirstPlan()
while plan != null do
  schedule ← Scheduler.getSchedule(plan, makespan) {B-a-B on makespan}
  if schedule.getMakespan() < makespan then
    makespan ← schedule.getMakespan() {better schedule found}
  end if
  Planner.getNextPlan(makespan) {next feasible plan with cut constraint}
end while
```

Solving approach is based on **Constraint Programming** technique, used in artificial intelligence and operations research, where we search for an assignment of given variables from their domains, in such a way that all constraints are simultaneously satisfied and value of an objective function is optimal.

Planning stage

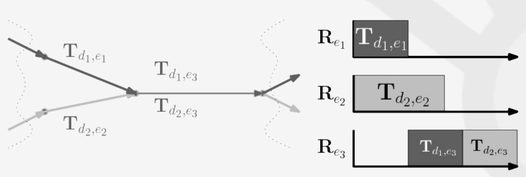
Link-based approach:

• The essential idea for this principle is to use one decision variable for each demand and link of the network (edge in a graph) - $\mathbf{X_{de}}$, denoting whether demand **d** is routed (value **1**) over the edge **e** of the network or not (value **0**).

• Mathematical constraints (similar to Kirchhoff's circuit laws) ensure that if all decision variables have assigned values the resulting configuration contains transfer paths.

Scheduling stage

• For each link **e** of the graph that will be used by at least one file demand, we introduce a unique **unary** resource $\mathbf{R_e}$. Similarly, for each demand and its selected links (defining a transfer path) we introduce a set of tasks **T**.



Inside the model

Computational complexity of the problem? Solving the second (scheduling) stage is **strongly NP-hard.** A possible polynomial-time reduction transforms an instance of $\mathbf{J3|p_{ij}=1|C_{max}}$, a 3-machine unit-time Job-shop scheduling problem (JSSP).

Therefore, a clever branching strategy (considering a search tree) and reduction of a space exploration is a key ingredient of the constraint approach.

Imposing time into planning phase

• Precedence constraints use non-decision positive integer variables $\mathbf{P_{de}}$ representing possible start times of transfer for demand **d** over edge **e**.

$$\forall d \in \mathbf{D}\ \forall n \in \mathbf{N}: \sum_{e \in \mathbf{IN}(n)} X_{de} \cdot (P_{de} + dur_{de}) \leq \sum_{e \in \mathbf{OUT}(n)} X_{de} \cdot P_{de} \quad (1)$$

• If we have some upper bound for the makespan (typically obtained as the best solution from the previous iteration of planning and scheduling) we can restrict plans in next iterations by constraint (3) where $\mathbf{SP_e}$ stands for the value of the shortest path from the ending site of **e** to **dest**.

$$\min_{f \in \mathbf{IN}(start)} (P_{df} + dur_{df}) \leq P_{de} \quad (2)$$

$$\forall e \in \mathbf{E}: \min_{d \in \mathbf{D}}(P_{de}) + \sum_{d \in \mathbf{D}} X_{de} \cdot dur_{de} + SP_e < makespan \quad (3)$$

Studied enhancements

• Symmetry breaking •

Detecting and breaking variable symmetries is one of the common techniques for reducing the search space. This is frequently done by adding variable symmetry breaking constraints that can be expressed easily and propagated efficiently using ordering.

• Implied constraints •

The usual purpose of filtering techniques is to remove some local inconsistencies and thus delete some regions of search space that do not contain any solution. By removing inconsistent values from variable domains the efficiency of the search algorithms is improved.

• Search heuristics •

Several combinations of variable selection and value iteration heuristics for both phases were tested. We mention:

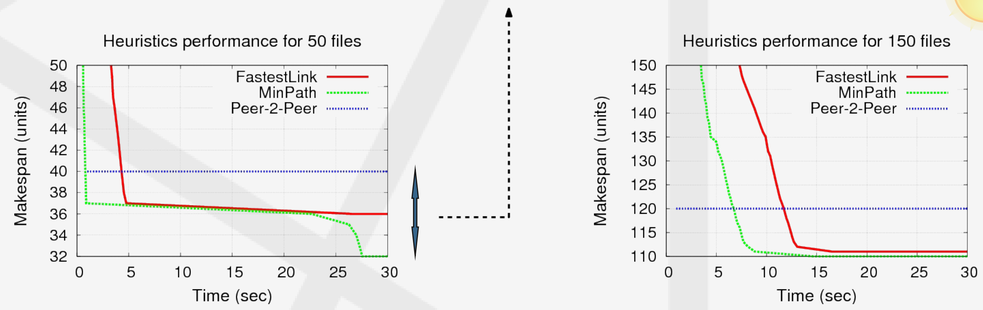
• ***FastestLink***
At the decision point from unassigned variables of demand **d**, the selected $\mathbf{X_{de}}$ corresponds to the fastest link.

• ***MinPath***
Instantiate first variable $\mathbf{X_{de}}$ such that the following value is minimal:

$$inf\ \mathbf{P_{de}} + \mathbf{dur_{de}} + \mathbf{SP_e}$$

For implementation of the solver we use **Choco**, a Java based library for constraint programming. Simulated experiments are designed to focus on evaluation of proposed alternatives of the model and detecting the most suitable combination of search heuristics. Understanding the performance and limitation of the studied techniques in a simulated real-like environment is a necessary step prior to further software deployment.

~ 20% shorter makespan



The performance of the *FastestLink* and *MinPath* heuristics and a *Peer-2-Peer* model was studied. Above graphs show that the convergence of the *MinPath* heuristic is faster than *FastestLink* and both achieve better makespan than P2P.

Conclusions

We tackled the complex problem of efficient data movements on the network within a multi-site environment. The problem itself arises from the **real-life needs** of the running nuclear physics experiment *STAR* and its **peta-scale requirements** for data storage and computational power as well.

Two stage constraint model, coupling **planning** and **scheduling** phase for data transfers to the single destination is presented. The **complexity** of the problem is introduced and several techniques for **pruning the search space**, like symmetry breaking, domain filtering, or cutting constraints, are presented. We propose and implement several **search heuristics** (FastestLink, MinPath) for both stages and perform sets of experiments with **realistic data input** for evaluating their applicability.

Constraint-based model and usage of **declarative type of programming** offers straightforward ways of representing many real life restrictions which are also **less vulnerable** to software coding errors in an ever expanding framework.

Comparison of the results and **trade-off** between the schedule of a constraint solver and a Peer-2-Peer simulator is promising and guarantees that continuation along this path will bring **improvements over the current techniques** to the community. In the nearest future we want to concentrate on the integration of the solver with real data transfer back-ends, consequently execute tests in the real environment.

CHEP 2009, Prague

E-mail address: michal.zerola@ujf.cas.cz