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

Michal Zerola for the STAR collaboration
Jérôme Lauret, Roman Barták and Michal Šumbera

michal.zerola@ujf.cas.cz

NPI, AS CR

ACAT 2008, November 4, 2008

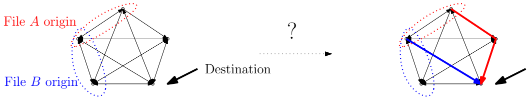
Outline of the problem

Challenges

How to tranfer 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 SE and CE** aspects for distributing data on the grid for further processing?

We concentrate on the first (most practical and of immediate need) question. On the grid:

- files are often replicated - available at multiple sites
- links diverse in bandwidth or QoS



We will present a Constraint Programming approach for solving this problem.

Introduction to Constraint Programming

A Constraint Satisfaction Problem (CSP) consists of:

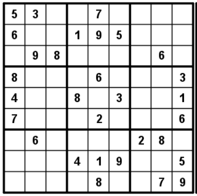
- a set of **variables** $X = \{x_1, ..., x_n\}$,
- for each variable x_i, a finite set D_i of possible values (its **domain**),
- and a set of **constraints** restricting the values that the variables can simultaneously take.

A solution to a CSP is an assignment of a value from its domain to every variable, in such a way that every constraint is satisfied. We may want to find:

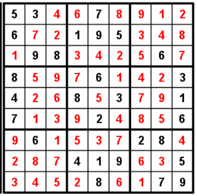
- any solution
- all solutions
- an optimal solution (defined by some objective function)

Example - Sudoku

- logic-based number-placement puzzle
- the objective is to fill a 9×9 grid so that each column, each row, and each of the nine 3×3 boxes contains the digits from 1 to 9 **only one time each**



5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9



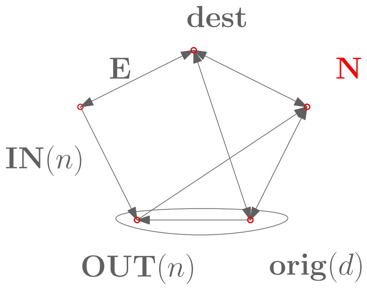
5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

How to model Sudoku as a CSP?

- **variable** with a **domain** equal to $\{1, \ldots, 9\}$ for each single cell
- **constraint** of inequality between all pairs of variables in each row, column, or 3×3 cell

From Toy problem to the real life

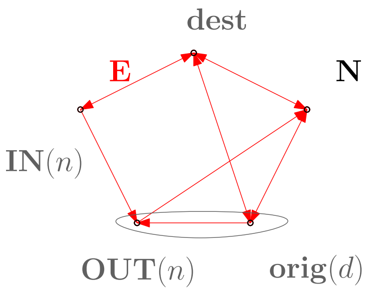
Notation:



D- set of demands

From Toy problem to the real life

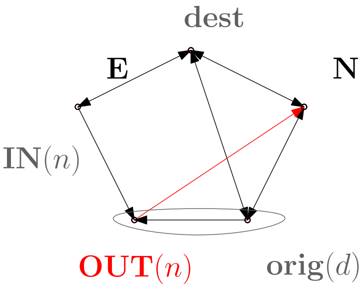
Notation:



D- set of demands

From Toy problem to the real life

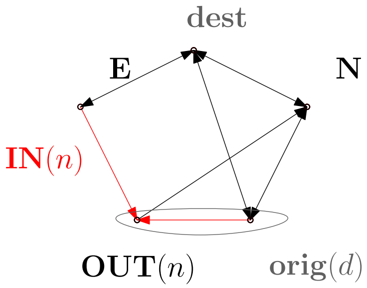
Notation:



D- set of demands

From Toy problem to the real life

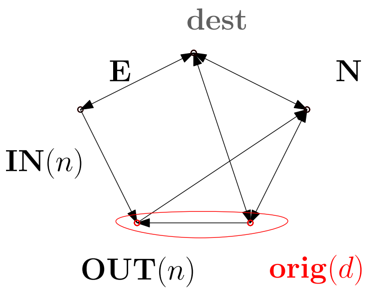
Notation:



$\mathbf{D}$- set of demands

From Toy problem to the real life

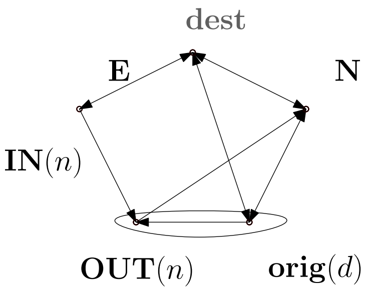
Notation:



$\mathbf{D}$- set of demands

From Toy problem to the real life

Notation:



$\mathbf{D}$- set of demands

Variables:

- Decision variable $\mathbf{X_{de}} = \begin{cases} 1, & d \text{ is routed over } e \\ 0, & d \text{ is not routed over } e \end{cases}$
- Integer variable $\mathbf{start_{de}}$ - start time of transfer d over e

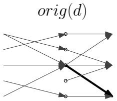
Modeling constraints

Objective function - **minimizing makespan** (latest finish time):

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

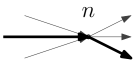
Extraction of constraints:

- for each file data transfer must start from exactly one of its origins



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

- if the file is transferred trough a given site, it must enter and leave site exactly once



$$\forall d \in \mathbf{D}, \forall n \notin \{orig(d) \cup dest(d)\} :$$

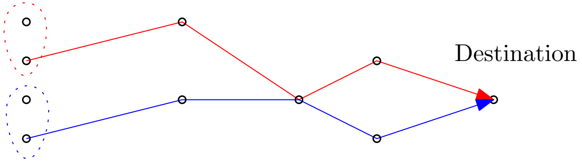
$$\sum_{e \in \mathbf{OUT}(n)} X_{de} \leq 1, \quad \sum_{e \in \mathbf{IN}(n)} X_{de} \leq 1, \quad \sum_{e \in \mathbf{OUT}(n)} X_{de} = \sum_{e \in \mathbf{IN}(n)} X_{de}$$

- $\cdots$ and many others

Solution finding principle

The solving is composed of tree search-like **iterations over two stages**:

1. choose a path for each demand from its origins to the destination (assigning X variables)



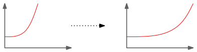
2. for each link schedule all transfers, as on disjunctive resource (assigning *start* variables)

Complexity

The second stage corresponds to job-shop scheduling that belongs to the $NP-hard$ class of problems.

Pruning the search space

We cannot escape exponential time growth but we can do the best to shift it, so our input instances will be solved fast.



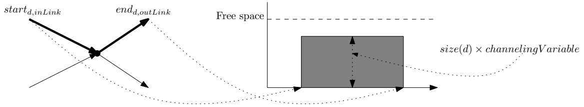
How to achieve it?

- **symmetry breaking** - don't explore configurations that are symmetrical (e.g. require ordering of the files from the same origin)
- **branch cutting**- don't explore transfer paths that cannot lead to the better schedule than we already have (too many file transfers per some link can make a lower bound estimate of the makespan)
- **heuristic** can lead to a fast (not necessarily optimum) solution, but "good enough"

. . . and other techniques, open for research

Additional real life constraints

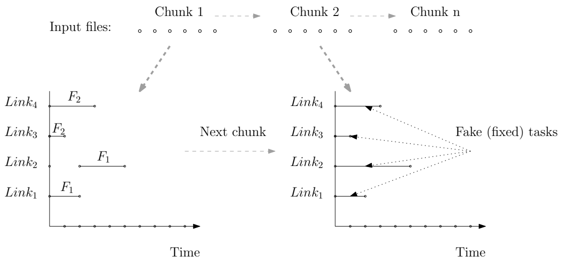
- sites have a **limited storage space**
  - easily modeled by *cumulative* constraint for each resource at a site



- several {in,out}coming links can have a **shared** bandwidth
  - easily modeled by *cumulative* constraint for each resource at a shared link
- **fair share** in a multiuser environment
  - scheduling by chunks, fair share corresponds to the selecting approach from waited demands
- network characteristic is not static but **fluctuates** in time
  - scheduling by chunks, sensors

Scheduling by chunks

We can split scheduling of the whole file set into parts (chunks) and create an optimal schedule for each part separately, while propagating previous results.



Benefits:

- speed
- self adaptation to the network
- fair-share (responsiveness)

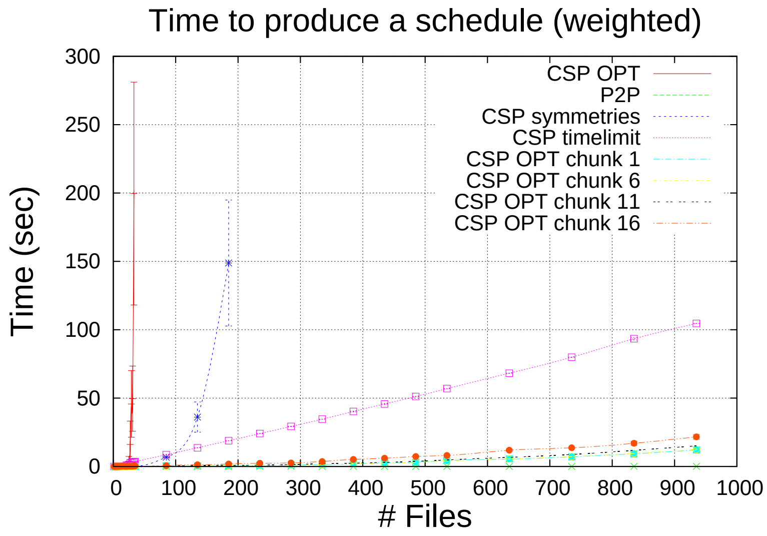
Implementation

We choosed Java library for implementation of the scheduler. The main heart of the solver is inside an open-source **Choco** library designed for CSP and CP.

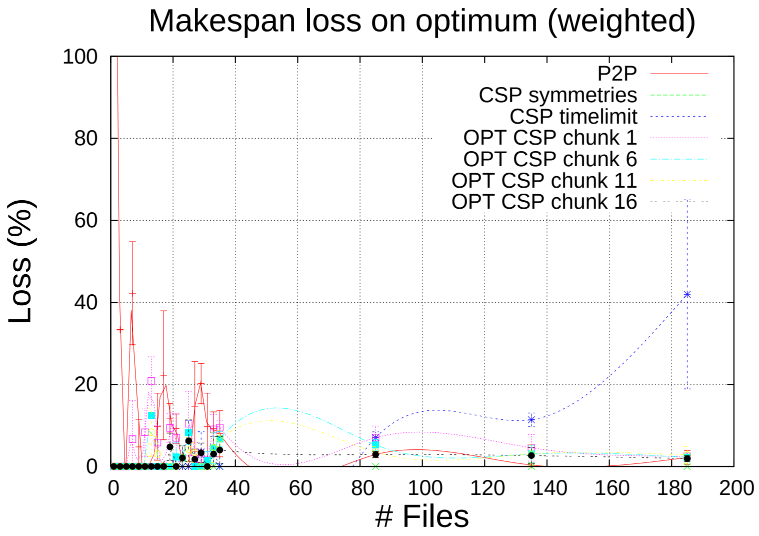
http://sourceforge.net/projects/choco/

- we implemented a simulator scheduling file transfers generated by feeders
- feeder generates file distribution with file origins:
  - distinct (file is available only on 1 site)
  - weighted (realistic probability function)
  - shared (file is available on each site)
- performance is compared to the Peer-2-Peer system

Results (1)

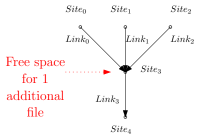

Time to produce a schedule (weighted)
Time (sec)
# Files
0
50
100
150
200
250
300
0
100
200
300
400
500
600
700
800
900
1000
CSP OPT
P2P
CSP symmetries
CSP timelimit
CSP OPT chunk 1
CSP OPT chunk 6
CSP OPT chunk 11
CSP OPT chunk 16

Results (2)



Makespan loss on optimum (weighted)
Loss (%)
# Files
P2P
CSP symmetries
CSP timelimit
OPT CSP chunk 1
OPT CSP chunk 6
OPT CSP chunk 11
OPT CSP chunk 16
0
20
40
60
80
100
0
20
40
60
80
100
120
140
160
180
200

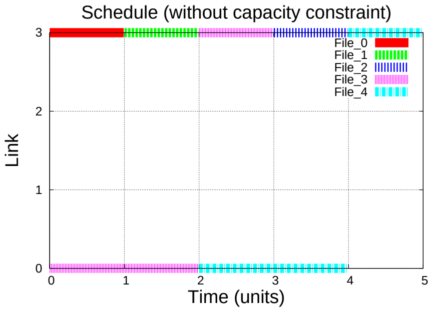
Effect of the capacity constraint

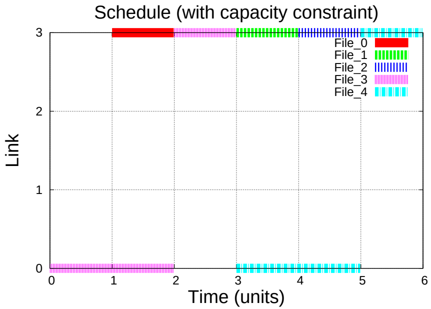


$orig(File_0) = \{Site_0, Site_3\}$
$orig(File_1) = \{Site_3\}$
$orig(File_2) = \{Site_0, Site_3\}$
$orig(File_3) = \{Site_0\}$
$orig(File_4) = \{Site_0\}$

$slowdown(Link_0) = 2$
$slowdown(Link_1) = 2$
$slowdown(Link_2) = 2$
$slowdown(Link_3) = 1$

Gantt charts:





Conclusions and future work

Constraint programming seem most suitable as:

- it allows to express in human terms real life problems
- it has a declarative character that allows easier further extensions

Further work would need to include:

- Enhancements for the 2^{nd} stage scheduling phase implementing algorithms from the job-shop problem field
- Testing several heuristics
- Move from simulation to the real life environment

Our first tests show promising results for single destination (site) data-movements.

- Out-performs p2p
- Performs well with planning-by-pieces (chunks)
- Hence, scales to large number of files