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

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

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

[2]Brookhaven National Laboratory, USA

[3]Czech Technical University in Prague, Czech Republic

[4]Masaryk University, Czech Republic

d.i.makatun@gmail.com

Outline

Motivation

Previous work

It was shown that global planning of **data-transferring** over Grid can outperform well known heuristics (e.g. P2P, Xrootd reasoning). [a]

[a] Michal Zerola et al "One click dataset transfer: toward efficient coupling of distributed storage resources and CPUs", 2012 J. Phys.: Conf. Ser. 368 012022 doi:10.1088/1742-6596/368/1/012022

Extension

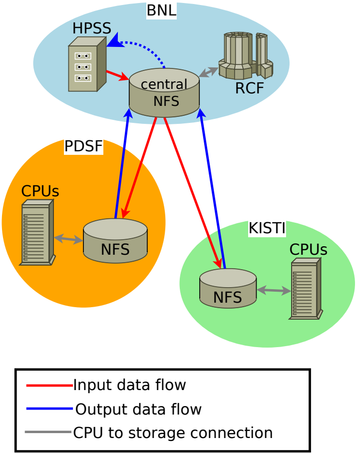
Global planning for **entire data-processing routine** in distributed environment.

Example of optimization

- What would be optimal:
  - ? Send a job to a site with slow connection **or** wait for a free slot at local site?
  - ? Access data remotely **or** transfer it before the job starts?
- Heuristics such as `[Pull a job when CPU slot is free]` will not give the answer.

Case 1: Data production. Planning remote site usage.

- RAW data is located at BNL.
- Computational resources are available at BNL and several remote sites.
- Long I/O overheads when accessing remotely stored data can reduce the applications CPUTime/WallTime ratio [6, 3].
- **How should we split a given dataset between sites to complete the processing faster?**



Manually adjust the number of remote jobs to meet the network throughput, **but** what if:

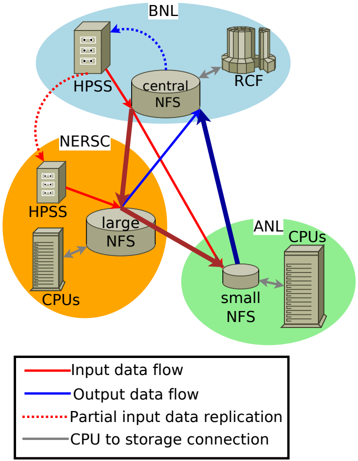
- More sites
- Changing network load

This should be automated.

Case 2: Data production. Optimization.

Consider entire GRID

- Several possible data sources.
- More complex network.
- Limited storage at sites.
- **How to distribute jobs by sites?**
- **Which file source to select?**
- **What is the optimal transfer path?**

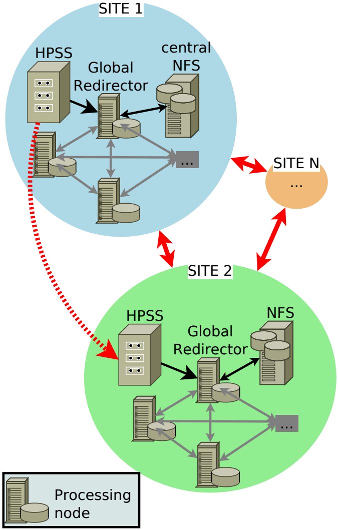


Example: data-production at ANL [6]

- ANL: many CPU's, but slow connection and small disk space.
- NERSC: fast connection, large disk.
- Optimization: Feed ANL from both BNL and NERCS sites.

Case 3: User analysis.

- Input data can be at any storage in the system.
- Data can be replicated.
- Each file can be requested by multiple jobs.
- 1 CPU per job.
- The size of output of analysis is negligible compared to input size.
- The processing time estimates are imprecise.
- **How to distribute the load?**
- **When and where to replicate the data?**



Goal

Create a global scheduler for Grid which will reason about:
1.data transferring, 2. CPU allocation, 3. data storage.

Optimization

- None of the resources (network links, data storages and CPUs) are over-saturated at any moment of time.
- The jobs are executed where the data is pre-placed.
- No excessive transfers or data replication.
- Minimal overall makespan for a given set of tasks.

Constraint programming with its techniques for scheduling, planning and optimization is a natural choice.

What is Constrain Programming?

Constraint programming is a form of declarative programming.
Widely used in: scheduling, logistics, network planning, vehicle routing, production optimization, etc.

Model

- Parameters
  constants
- Variables
  bool, int, float
- Domains
  set of values
- Constraints
  math expressions
  - Core
  - Redundant

Solution

Assign values to variables to satisfy constraints.

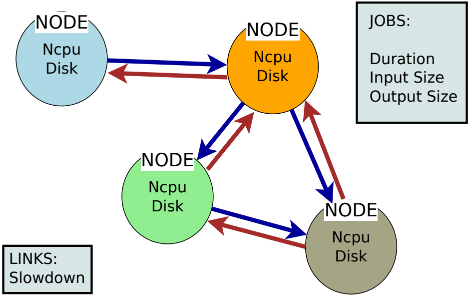
Optimal solution

Maximize/minimize target function.

Search

- Algorithm:
  e.g. backtracking.
- ◦ Variable order.
- ◦ Value order.
- \+ Consistency Techniques.
- Complete vs Incomplete search.

Data-production problem: Input.



Assumptions

In previous work [2] it was proved that:

- There is advantage to plan and schedule jobs by chunks.
  - + More adaptability to changing environment.
  - + Faster plan creation.
- The network links can be considered as unary resources: one file-transfer at a time over link.

Data-production problem: Variables.

Input parameters:

- Nodes c
  - CPUs: $N_{CPU}(c)$
  - Disk space: $Disk(c)$
- Links l
  - Starting Node.
  - End node.
  - Slowdown = 1 / Bandwidth
- Jobs j
  - Duration.
  - Input size.
  - Output size.
  - Input source node(s).
  - Output destination node(s).

Domain variables:

- $Y_{jc} \in \{0, 1\}$ job j processed at node c.
- $X_{fl} \in \{0, 1\}$ file f transferred over link l.
- Js_j start time of job j.
- Ts_{fl} start time of transfer of file f over link l.

Dependent on above

- Fs_{fc} start time of file f placement at node c.
- $Fdur_{fc}$ duration of file f placement at node c.

Solving procedure overview.

1. Initialization Stage. Estimate *TimeLimit*.
2. Planning Stage. Instantiate a part of domain variables with the help of simplified constraints.
   a. Assign jobs to computational nodes.
   b. Select transfer paths for input and output files.
   c. Additional constraints: load balance, etc.
   d. Find a solution for the sub-problem.
3. Scheduling stage: define start time for each operation.
   a. Constraints on order of operations.
   b. Cumulative constraints.
   c. Minimize target function: (e.g. **makespan**).

Planning stage (core constraints)

Each job processed exactly at one node:

$$\forall j \in J : \sum_{c \in C} Y_{jc} = 1$$

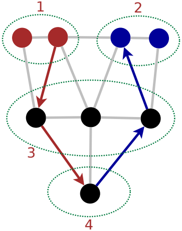
Target function T_{est} - estimated makespan.

For each node c: $T_{Processing} + T_{InputTransfer} + T_{OutputTransfer} \leq T_{est}$

Path selection

File can be transferred from/to each node at most once.

1. Transfer input file from sources over 1 link.
2. Transfer output to final destination over 1 link.
3. Intermediate node: If $\exists$ incoming transfer $\Leftrightarrow \exists$ outgoing transfer.
4. Selected processing node: 1 incoming input transfer, 1 outgoing input transfer.



Scheduling Stage: order of tasks.

Outgoing transfer starts after the incomming one is finished:

Ts- transfer start. $\forall f \in F, \forall c \in IntermediateNode$

$$Ts_{fl_{out}} \geq Ts_{fl_{in}} + Size(f) \cdot Slowdown(l_{in})$$

Jobs starts after the input file transfer is finished

Js-job start. $\forall j \in J, l \in L, f = InputFile(j)$

$$Js_j \geq Ts_{fl} + InputSize(j) \cdot Slowdown(l)$$

Output file is transferred after the job is finished

$\forall j \in J, l \in L, f = OutputFile(j)$

$$Js_j + Dur(j) \leq Ts_{fl}$$

Scheduling Stage: data placement

Space reservation at destination node is made when transfer starts

Fs - start of file placement. l is link to c: $Fs_{fc} = Ts_{fl}$

File can be deleted from start node of a link after the transfer

$Fdur$- duration of file placement.

$$Fs_{fc} + Fdur_{fc} = Ts_{fl} + Size(f) \cdot Slowdown(l)$$

At selected processing node

When a job starts space for output is reserved $f = OutputFile(j)$:

$$Fs_{fc} = Js_j$$

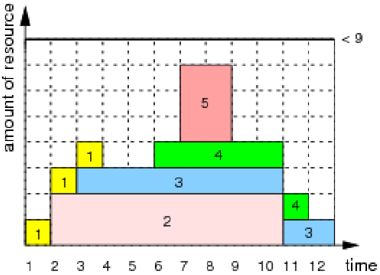
When a job finishes it's input file can be deleted $f = InputFile(j)$:

$$Fs_{fc} + Fdur_{fc} = Js_j + Duration(j)$$

Scheduling Stage: cumulative constraints.

cumulative

Requires that a set of tasks given by **start times** s, **durations** d, and **resource usage** r, never require more than a **resource limit** b at any time.



Task	Start	Duration	Usage	Limit
Job	Js_{jc}	$Duration(j)$	1	N_{CPU}(c)
Transfer	Ts_{fl}	$Size(f) \cdot Slowdown(l)$	1	1
File placement	Fs_{fc}	$Fdur_{fc}$	$Size(f)$	$Disk(c)$

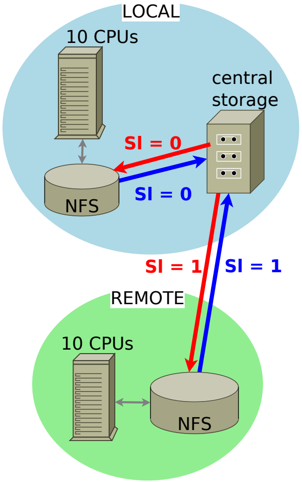
Testing simulations: problem setup.

Input for simulations

- Job input size: random 1..20
- Job duration: 1..48
- Job output Size: 1..22
- Input/output size and duration are proportional.

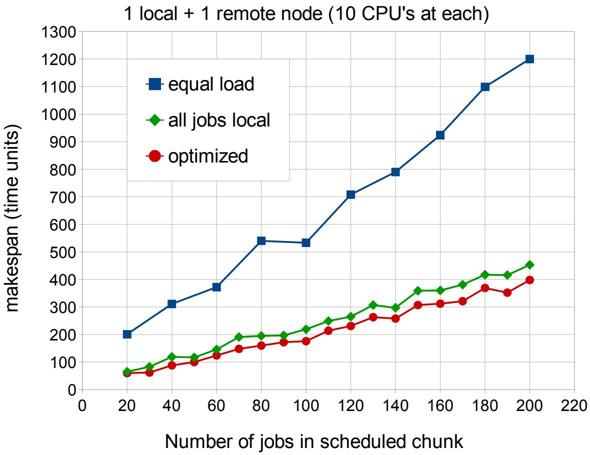
Tested algorithms

- All jobs processed at LOCAL node by input order.
- Equal CPU load. Processed by input order.
- Optimized. Planner: minimize estimated makespan. Scheduler: minimize makespan.



Constraints for storage capacity are omitted.

Testing simulations: results.



equal load: +166% optimized: -15% of makespan compared to local processing

Conclusions

Conclusions

- Mathematical model (Constraint Satisfaction Problem) for scheduling of data-production over Grid was formulated.
- In simulated environment, where a remote site has the same CPU number as a local site, but data transfer overhead is comparable to job duration:
  - Maintaining **equal CPU load** at local and remote sites increases the makespan more then twice;
  - Scheduling with **consideration of transfer overhead** can reduce makespan by 15%.

  compared to **local only** processing.
- Proposed approach can provide **optimization** and **automatic adaptation** to fluctuating resources with no need for manual adjustment of work-flow at each site or tuning of heuristics.

Future plans

- Test on larger problems (more nodes, more CPUs, more links). Compare results to statistic logs.
- Implement a custom search heuristics for CSP.
- Improve search performance in order to enable online scheduling.

End.

Thank you for your attention.

Visegrad Fund

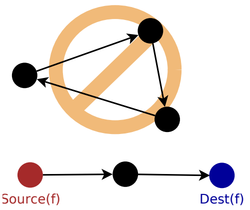
U.S. DEPARTMENT OF ENERGY

Backup 1. Link selection: loop elimination.

For each subset of k nodes C_k the number of transfers on internal links should be less then k.

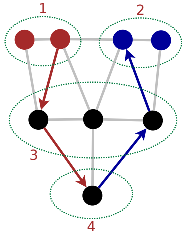
$$\forall f \in F, \forall k : 2 \leq k \leq \|C\|, \forall C_k \subset C$$

$$\sum_{l \in L : Begin(l) \in C_k \wedge End(l) \in C_k} x_{fl} < k$$



- If too many constraints => slows down the search.
- Work around: Use for $k = 2$ only. Remove the rest of cycles after the plan is generated [4].

Backup 2. Link selection.



File can be transferred from/to each node at most once:

$$\forall c \in C : \sum_{l \in inLinks(c)} X_{fl} \leq 1; \quad \sum_{l \in outLinks(c)} X_{fl} \leq 1;$$

1. Transfer input file from source: $\forall f \in InputFiles$:
$$\sum_{l \in linksFromSources} X_{fl} = 1; \quad \sum_{l \in linksToSources} X_{fl} = 0$$
2. Transfer output to final destination: $\forall f \in OutputFiles$:
$$\sum_{l \in linksToDest} X_{fl} = 1, \quad \sum_{l \in linksFromDest} X_{fl} = 0$$
3. At intermediate node: $$\sum_{l \in inLinks(c)} X_{fl} = \sum_{l \in outLinks(c)} X_{fl}$$
4. At selected processing node: $Y_{jc} = true, f_{in} = InputFile(j)$, $f_{out} = OutputFile(j)$ $$\sum_{l \in inLinks(c)} X_{f_{in}l} = 1; \quad \sum_{l \in outLinks(c)} X_{f_{in}l} = 0;$$
$$\sum_{l \in inLinks(c)} X_{f_{out}l} = 0; \quad \sum_{l \in outLinks(c)} X_{f_{out}l} = 1;$$

[1] Makatun D, Lauret J and Šumbera M 2013 Study of cache performance in distributed environment for data processing, *J. Phys.: Conf. Ser. ACAT Conf.* Peking 2013

[2] 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. Ser.* **368**

[3] Horký J, Lokajíček M and Peisar J 2013 Influence of Distributing a Tier-2 Data Storage on Physics Analysis *J. Phys.: Conf. Ser.* ACAT Conf. Peking

[4] Troubil P and Rudová H 2011 Integer Linear Programming Models for Media Streams Planning. *Lecture Notes in Management Science* **3** 509-522

[5] L. Betev, A. Gheata, M. Gheata, C. Grigoras and P. Hristov 2014 Performance optimisations for distributed analysis in ALICE *J. Phys. Conf. Ser.* **523** (2014) 012014.

[6] J. Balewski, J. Lauret, D. Olson, I. Sakrejda, D. Arkhipkin, J. Bresnahan, K. Keahey and J. Porter *et al.*, Offloading peak processing to virtual farm by STAR experiment at RHIC 2012 *J. Phys. Conf. Ser.* **368** (2012) 012011.