Distributed Data Processing for High Energy Physics

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

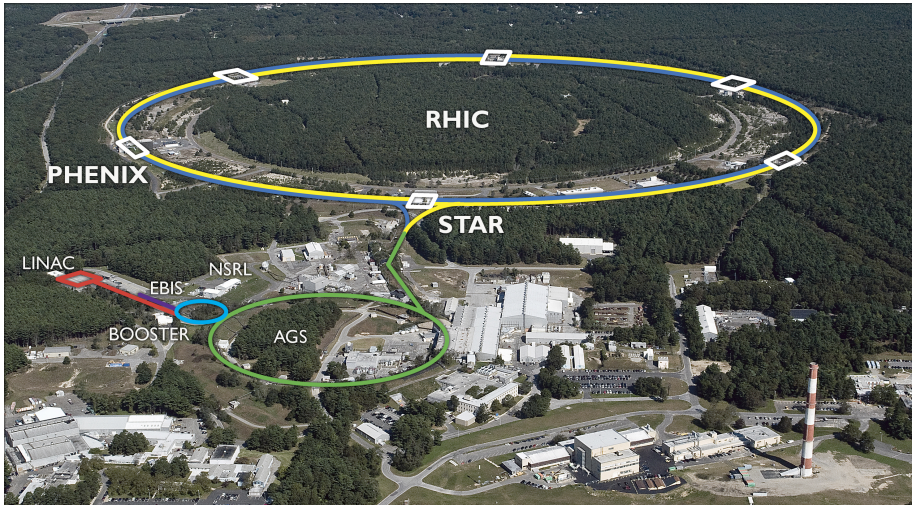
[4]Faculty of Informatics, Masaryk University, Czech Republic

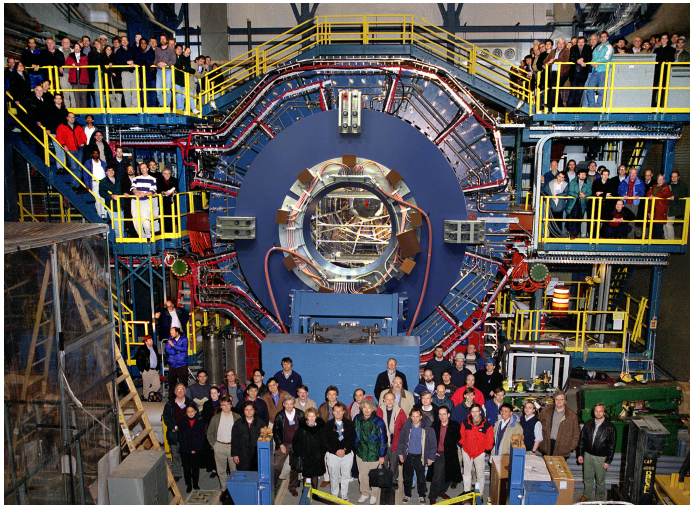
d.i.makatun@gmail.com

June 21, 2015

Outline

Computations in HEP: what do we compute?

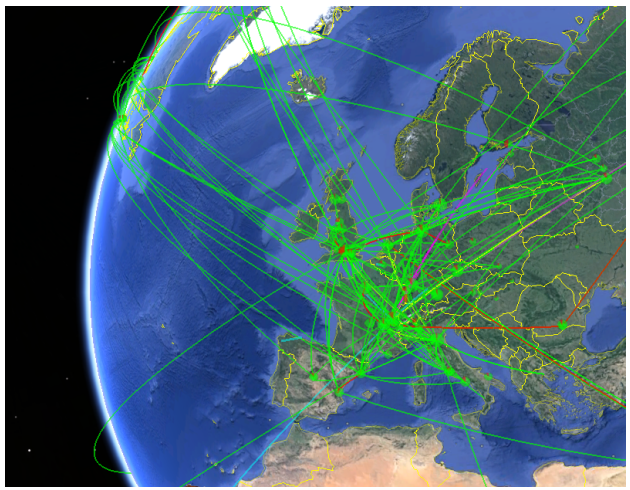


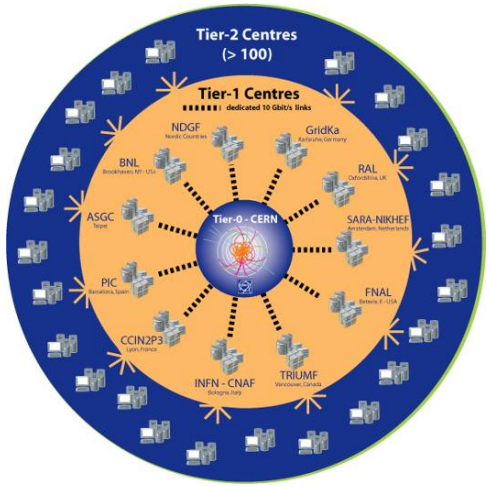


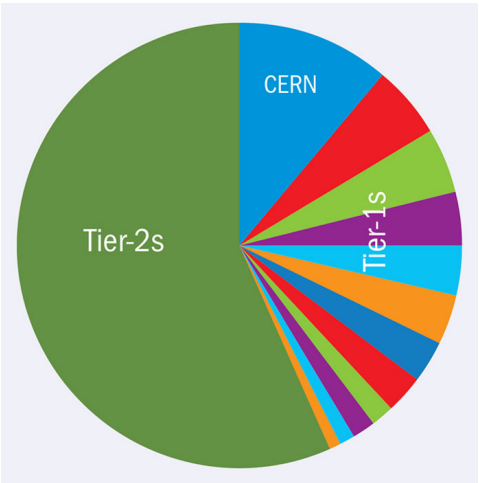
- Brookhaven National Laboratory (**BNL**) Long Island, NY, USA
- Relativistic Heavy Ion Collider (**RHIC**). In Gold-Gold ion collisions a quark-gluon plasma is created to study the primordial form of matter that existed in the universe shortly after the Big Bang.
- Solenoid Tracker at RHIC (**STAR**). Collisions occur millions of times per second. Events of size 200 MB are processed at input rates up to 100Hz. Output data rate is $\sim$ 30 MB/sec.

Computations in HEP: how do we compute?

Data Production: The raw output data is processed to reconstruct events. ($\sim$ ones)
User Analysis: Then the reconstructed events are analyzed by scientists to discover new physics. (Each file many times)
All the data: raw, reconstructed and analysis output are stored.
31 PB of data stored on tape, $\sim$ 12 000 jobs running simultaneously (at RCF only).







Previous work and motivation.

RIFT: **R**easoner for **I**ntelligent **F**ile **T**ransfer.

Efficient and controlled movement of replicated datasets within Grid to satisfy multiple requests in the shortest time. [1]

- Select between several data sources.
- Create optimal transfer paths, merge shared transfer paths of the same file.
- Schedule transfers on links.

For our class of problems it was shown that global planning of **data-transferring** over Grid can outperform well known heuristics (e.g. P2P, Xrootd reasoning)

Extension

Global planning for jobs coupled with transfers in distributed environment.

Example of decisions

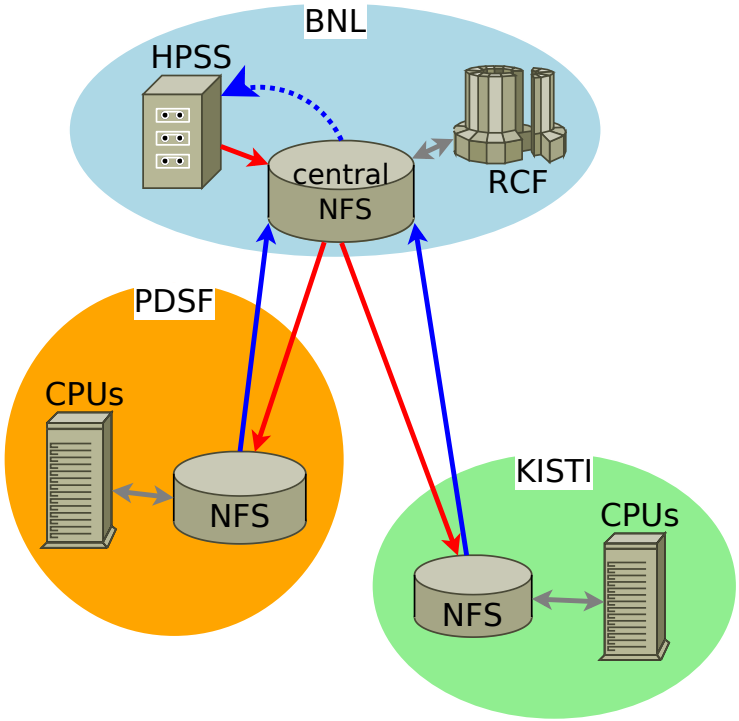
? Send a job to a site with slow connection **or** wait for a free slot at local site?

? 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.
- 1 job per file.
- 1 CPU per job.
- Input size $\approx$ Output size
- Output file has to be transferred back to BNL.
- **How should we distribute a given set of files 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.

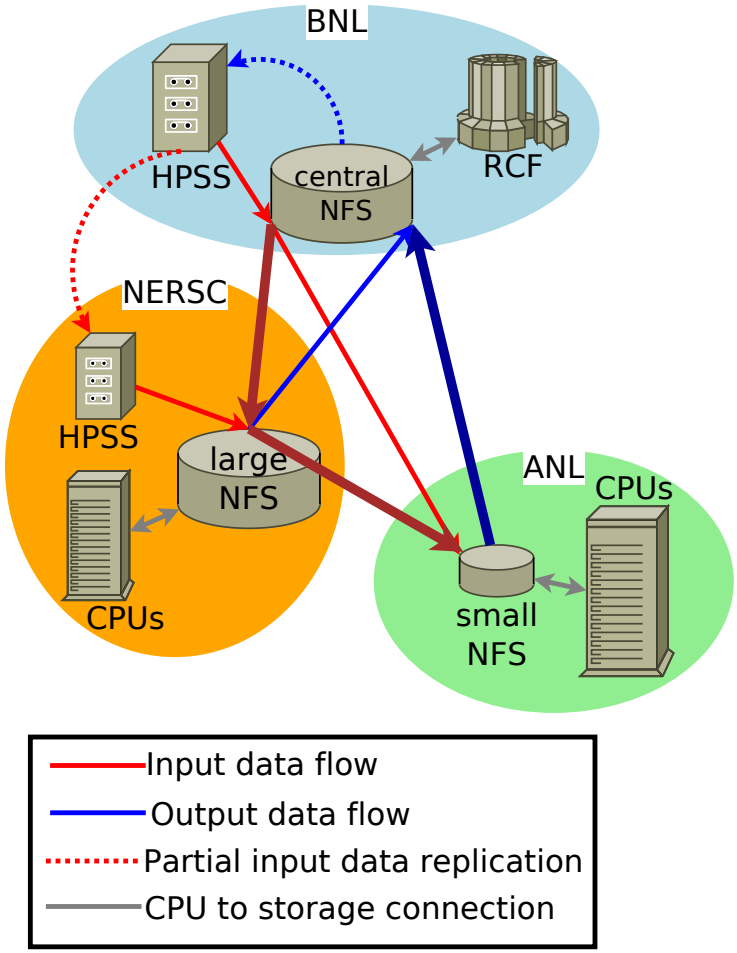
Case 2: Data production. Optimization.

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?**

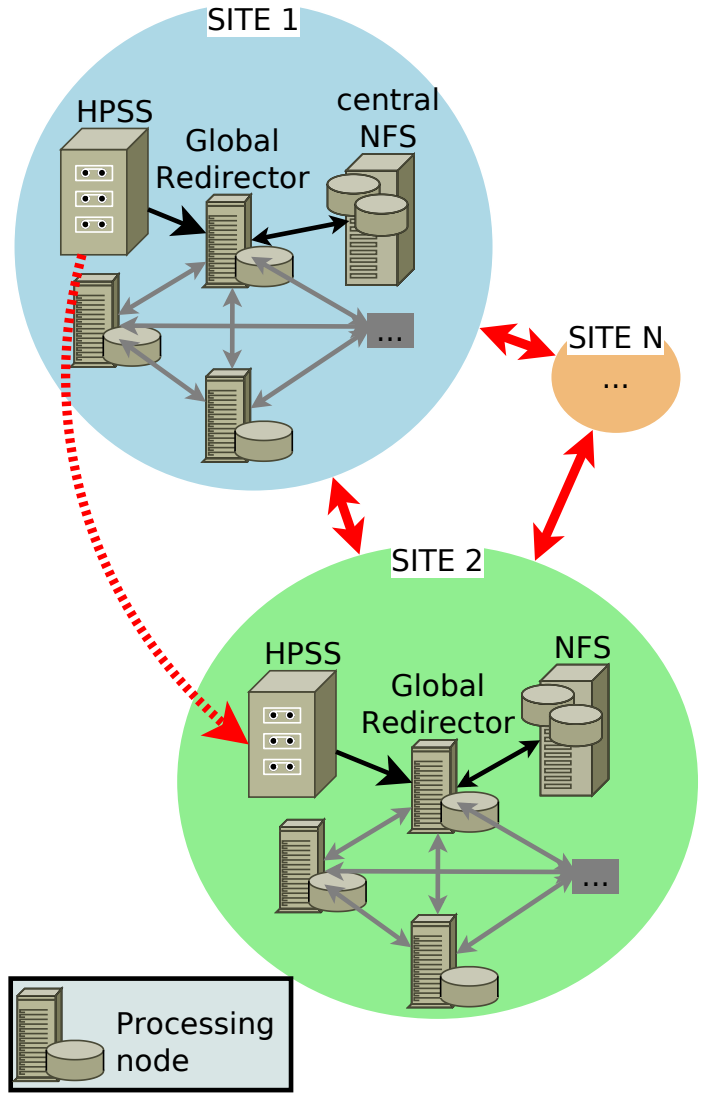
Example: data-production at ANL [2]

- 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.

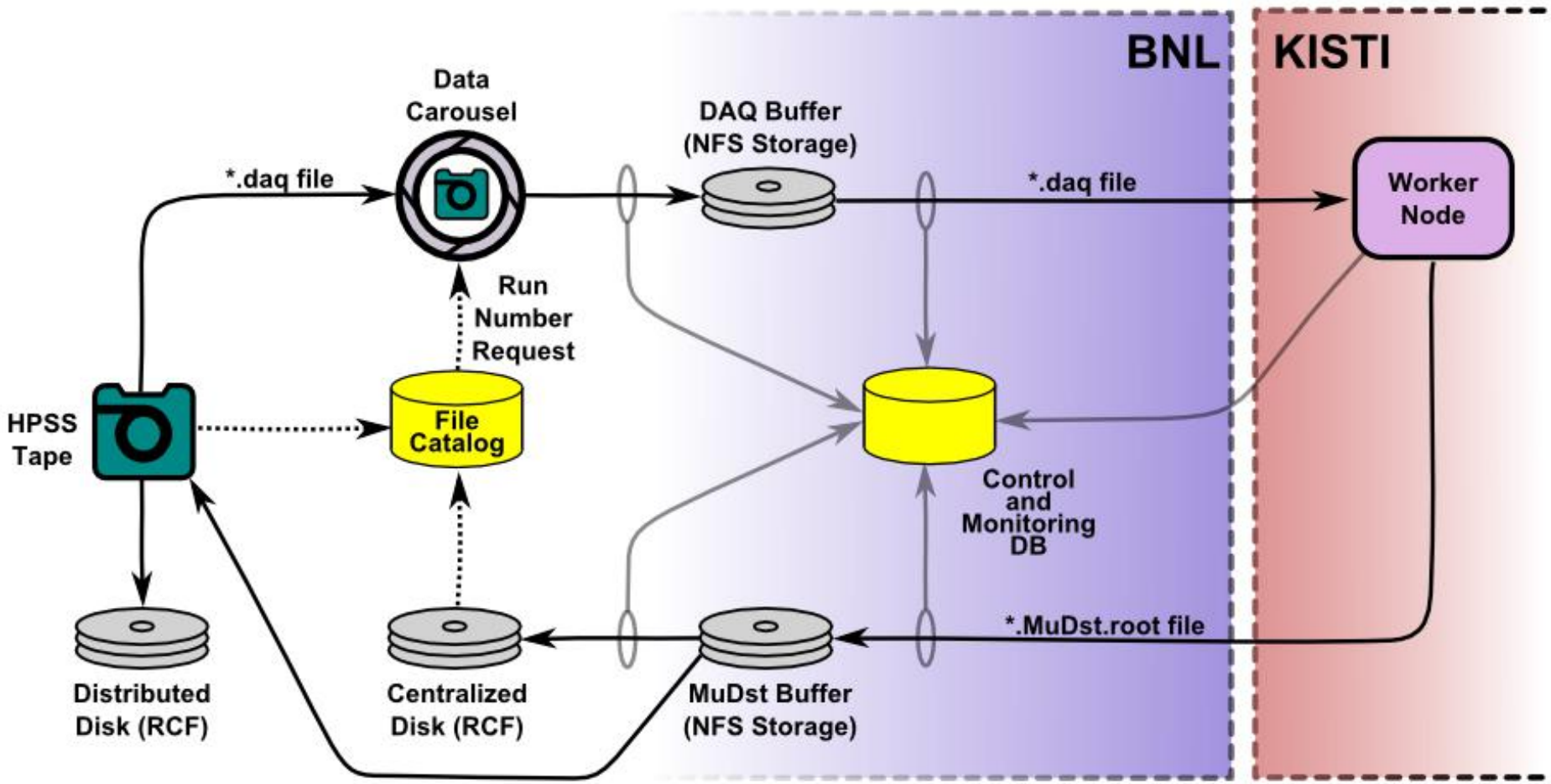


Case 3: User analysis.

- Many copies of files exist.
- 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?**



STAR: setup for data production at a remote site (KISTI)



- For better efficiency an ad-hock setup is used. [3]

Existing solutions (simulations)

Bandwidth-Centric Allocation of Independent Tasks on Heterogeneous Platforms [4]

- Exact solution for maximum steady-state throughput.
- Grid network is modeled as a tree (no alternative paths). Single source/destination. Input path = Output path. Equal size of jobs/files.

XSufferage [5]

- Considers I/O transfer latency. Assigns jobs to hosts based on $Sufferage = SecondBestEstimatedMakespan - BestEstimatedMakespan$
- No path/source selection or transfer planning. Simplified network model. No storage model.

Storage Affinity [6]

- XSufferage + job replication: Executes copies of the same job at several clusters concurrently.
- Simplified network model. CPU waste $\sim 25 - 60\%$.

Existing solutions (in use)

Batch System + Distributed Data Management System (Independent)

- PBS, Condor. [Pull a job from global queue]
- Xrootd, DPM. [Site which replies first is selected as a source]

Data Trains (For user analysis)

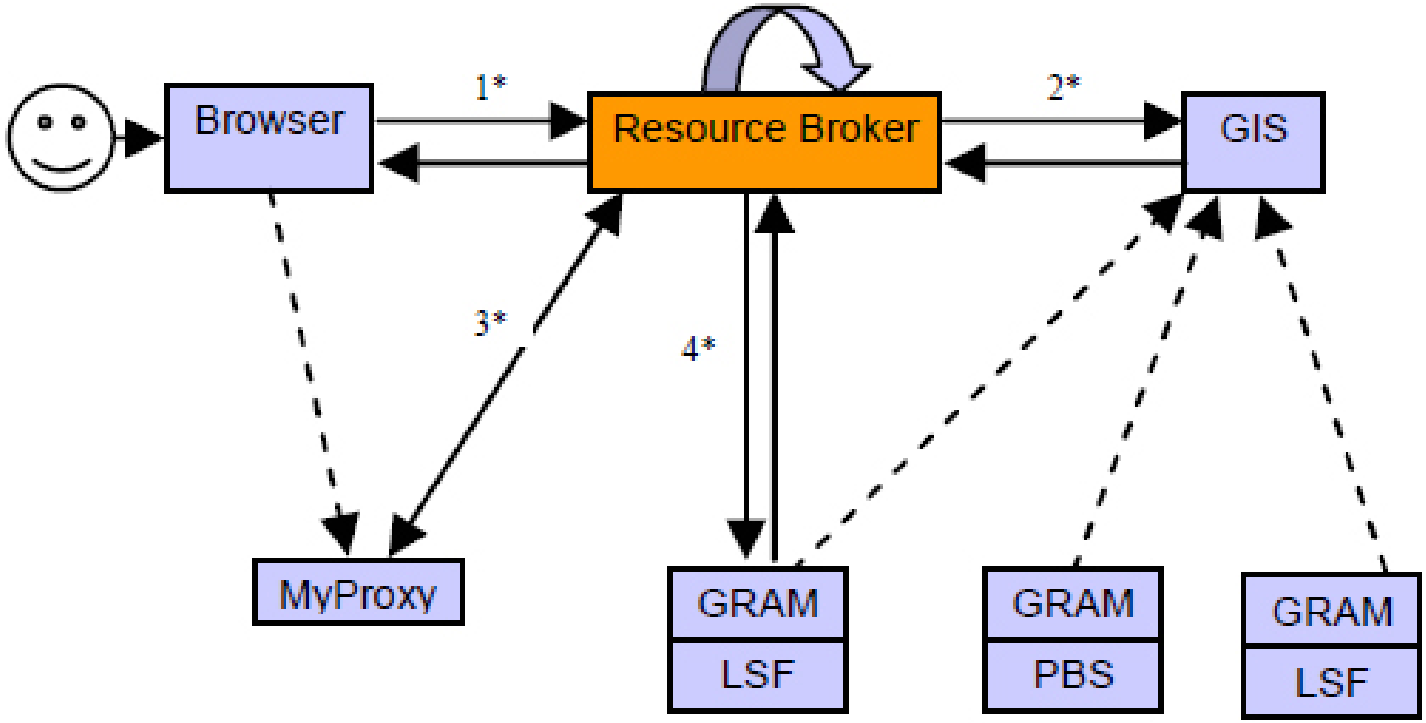
- Group jobs by input data $\longrightarrow$ preplace data $\longrightarrow$ start jobs simultaneously $\longrightarrow$ kill latest x% of jobs.
- Train runs periodically. ($\sim$ ones per day)
- Controlled by train operators.

Globus (Decoupling jobs and transfers)

- Sends jobs to data.
- Replicate most "popular" files. Relies on usage history.
- Where to replicate? When to replicate? No transfer planning.

Distributed resource management system. Architecture (Globus example) [7]

- The resource broker is acting as a middle tier between a user and the resources by doing resource matching and job submission for the user.



Data production problem

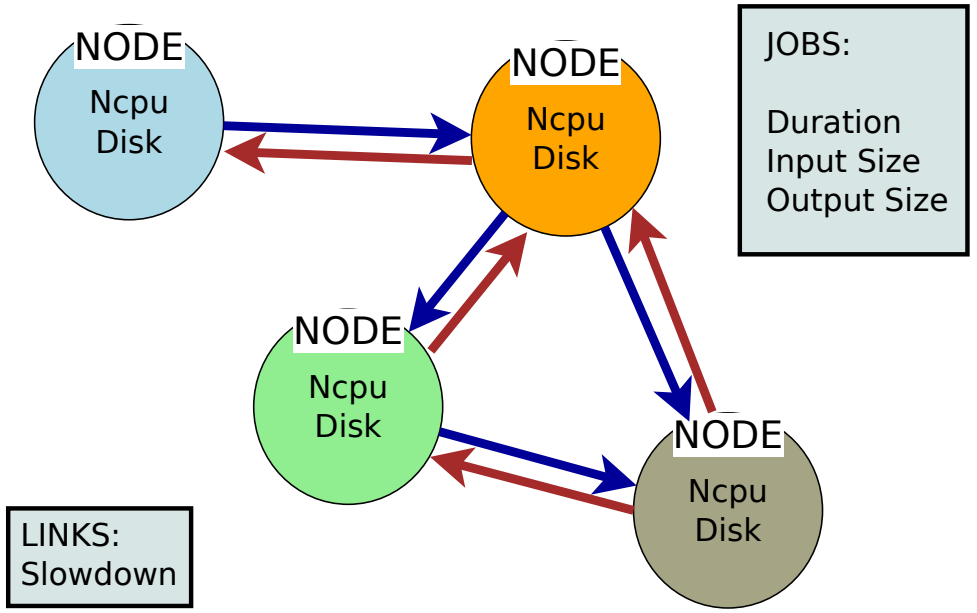
Create a global scheduler for Grid which will reason about:
1.data transferring, 2. CPU allocation, 3. data storage.

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.

To solve the problem we applied Constraint programming due to its techniques for scheduling, planning and optimization.

Data-production problem: Input.



Assumptions

In previous work [1] it was proved that:

- There is advantage to plan and schedule jobs by chunks (split the whole set by portions).
  - \+ 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 disk space reservation for file f at node c.
- $Fdur_{fc}$ duration of space reservation for file f 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: (job j, node c)

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

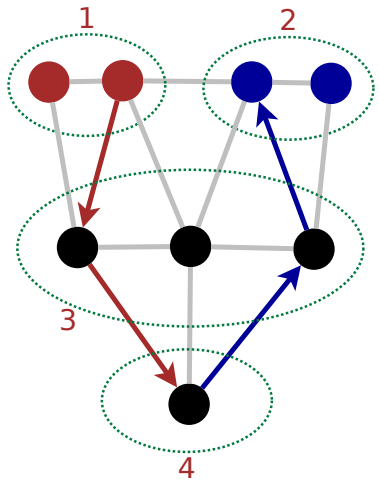
Target function T_{est} - estimated makespan.

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_{fl} transfer start of file f over a link l, c is a node.
$\forall f \in F, \forall c \in IntermediateNode$:

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

Jobs starts after the input file transfer is finished

Js_j start time of a job j. $\forall j \in J, l \in L, f = InputFile(j)$

$$Js_j \geq Ts_{fl} + Size(f) \cdot Slowdown(l)$$

Output file is transferred after the job is finished

$Dur(j)$ is duration of a job j. $\forall j \in J, l \in L, f = OutputFile(j)$

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

Scheduling Stage: data placement

Space reservation at destination node is made when transfer starts

Fs_{fc} - start of reservation for file f at node c, Ts_{fl} transfer start of file f over a link l to node c: $Fs_{fc} = Ts_{fl}$

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

$Fdur_{fc}$- duration of file f placement at node c, l is outgoing link from c

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

At selected processing node c

When a job j starts (Js_j) then space for output is reserved

$f = OutputFile(j)$: $Fs_{fc} = Js_j$

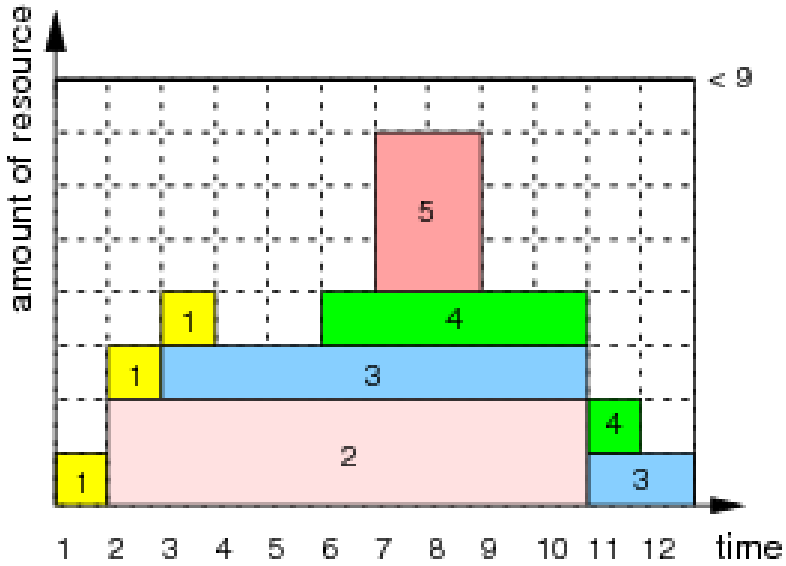
When a job finishes its input file can be deleted $f = InputFile(j)$:

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

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}	$Fdur_{fc}$	$Size(f)$	$Disk(c)$

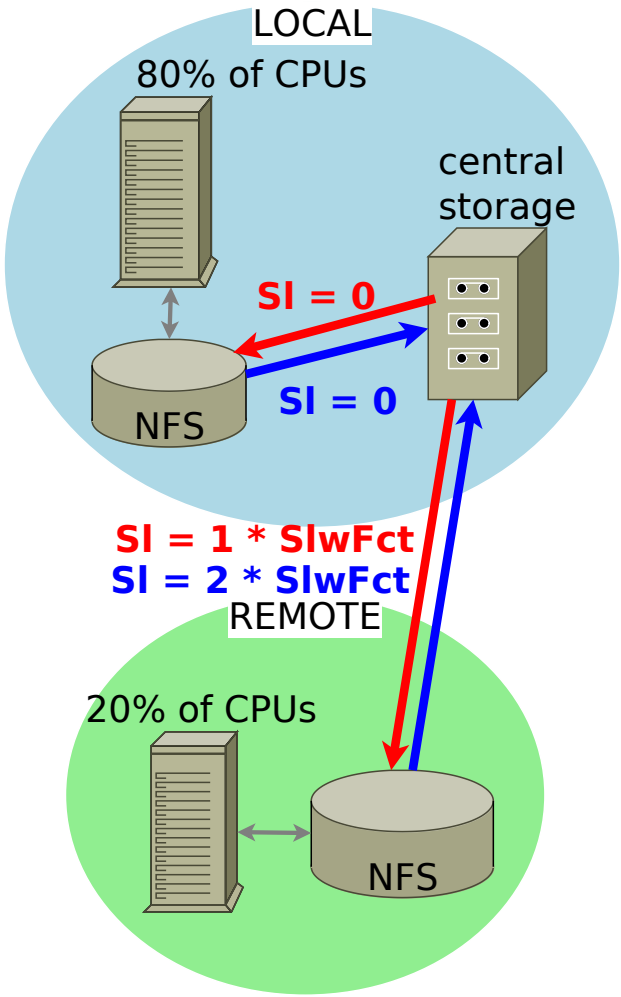
Simulations based on log data: problem setup.

Input for simulations

- Parameters of 2000 jobs taken from log files of data production for STAR at KISTI.
- Jobs scheduled by chunks of 200.
- Slowdown of the link to the remote site proportional to the **slowdown factor**.
- Makespan compared to local processing.

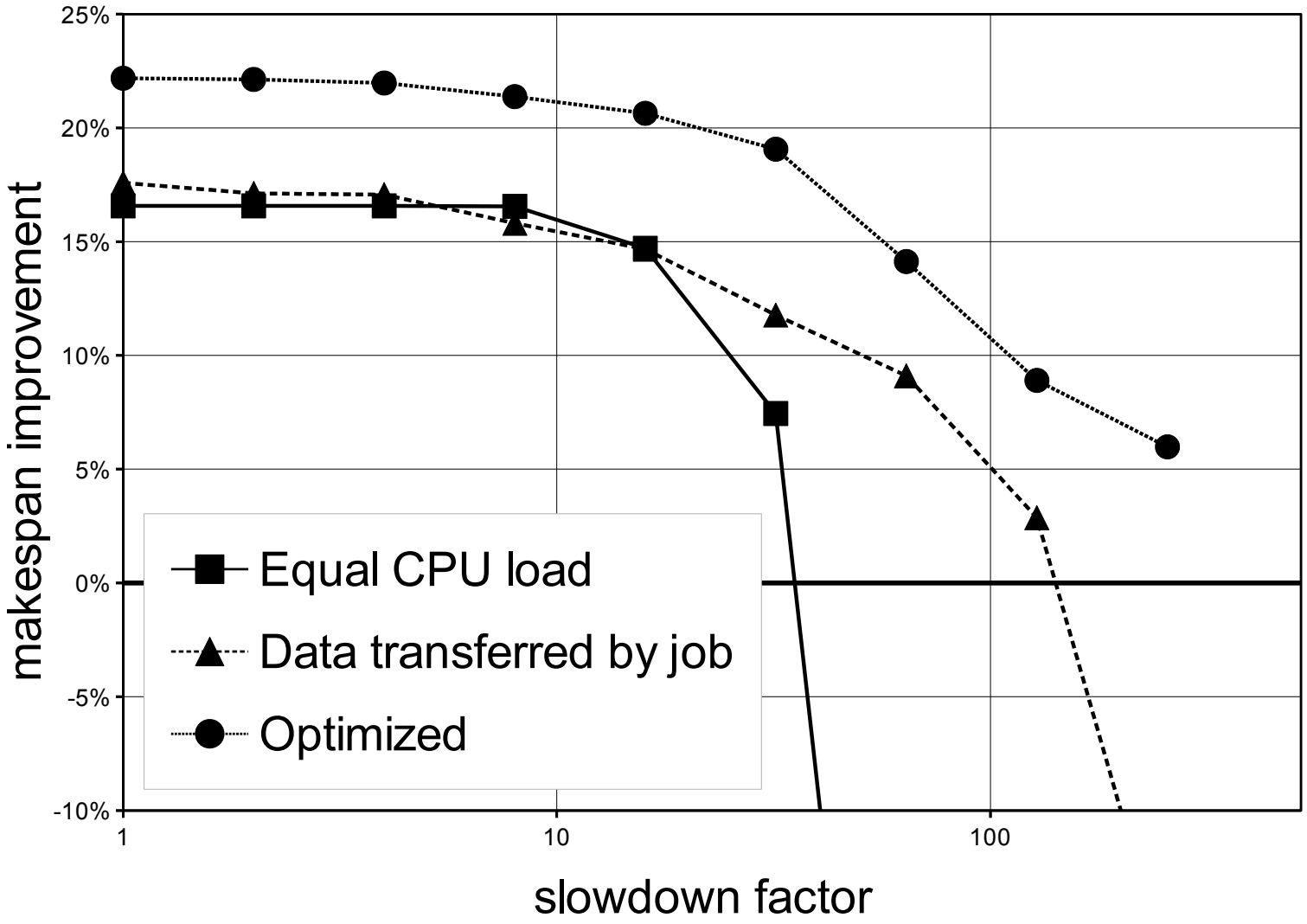
Tested algorithms

- Equal CPU load.
  Processed by input order.
- Data transferred by job.
  Processed by input order.
- Optimized.
  Planner: minimize estimated makespan.



Constraints for storage capacity are omitted.

Simulations based on log data: results.



Results of simulations

- 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.
- The simulations based on log files has shown that the proposed approach systematically provides a smaller makespan and adapts to the increase of transfer overheads better then the other simulated heuristics.
- 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.

Motivation

Drawbacks of CP model

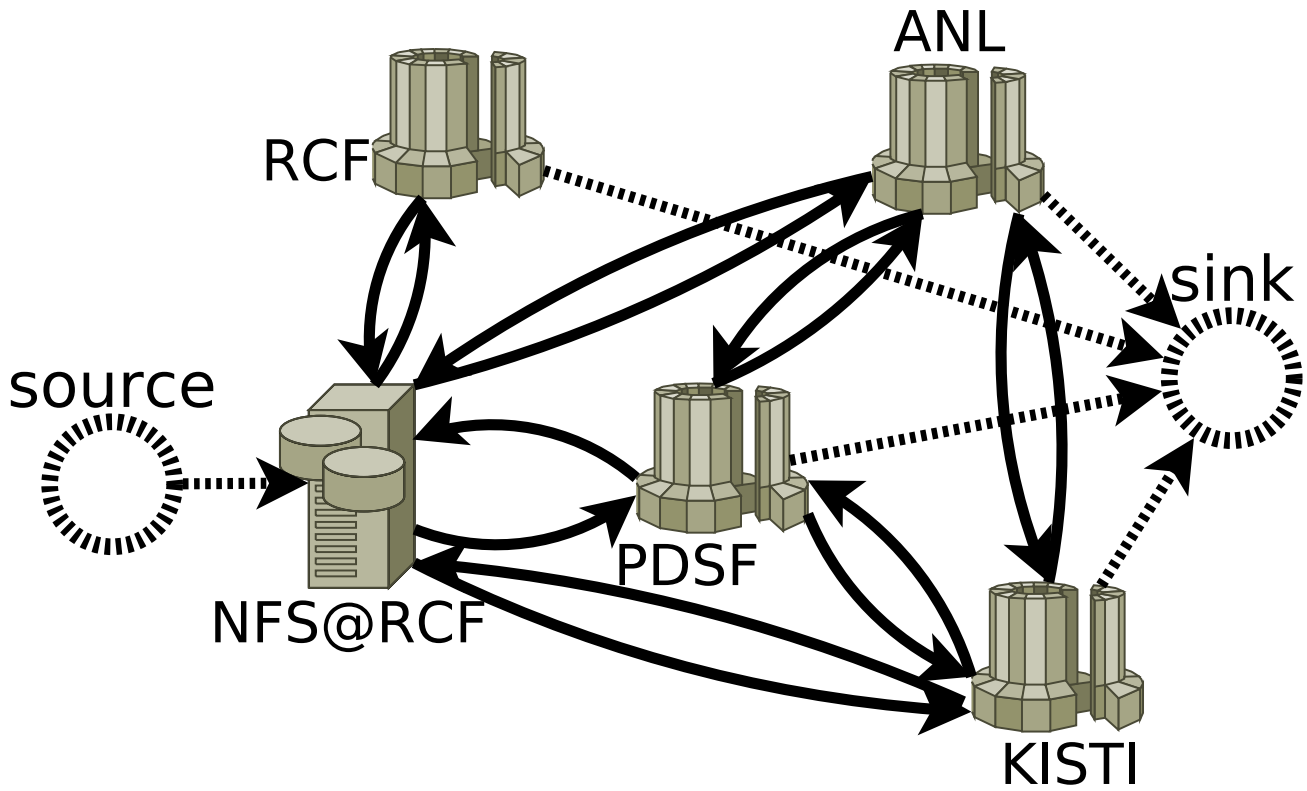
- Slow performance (NP-hard problem).
- Unnecessary calculations:
  - Path for each job.
  - Node selection for a particular job.
  - Order of jobs.
  - Makespan of a particular job.

Network flow model

- Similar problem solved: high-quality media streams planning [8].
- Idea: plan resource load only and then distribute particular jobs accordingly. Planning time interval ΔT, *Flow* - amount of data, link capacity - maximum bandwidth.
- Network flow maximization problem can be solved within polynomial time.

Input transfer planning: How much data can be transferred during the next planning interval ΔT?

Dummy edges - constraints on storage and CPUs at each site.



Output flow problem can be formulated similarly.

Solving procedure

Problems for input and output transfers can be solved independently under assumptions:

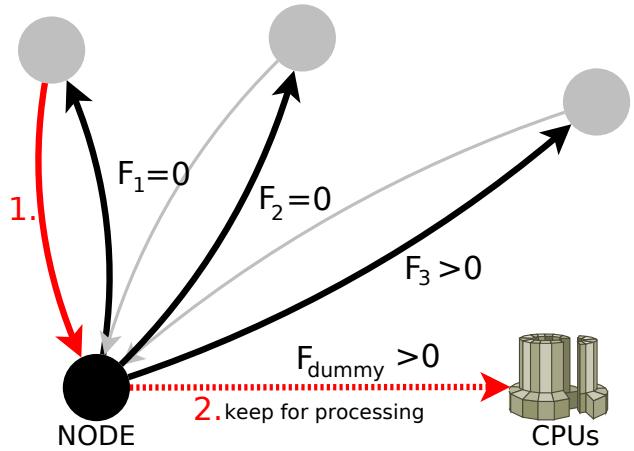
- Full-duplex links,
- In a steady state at each node
  $Processed\_input\_data = \beta \cdot created\_output\_data$, where $\beta \leq 1$.

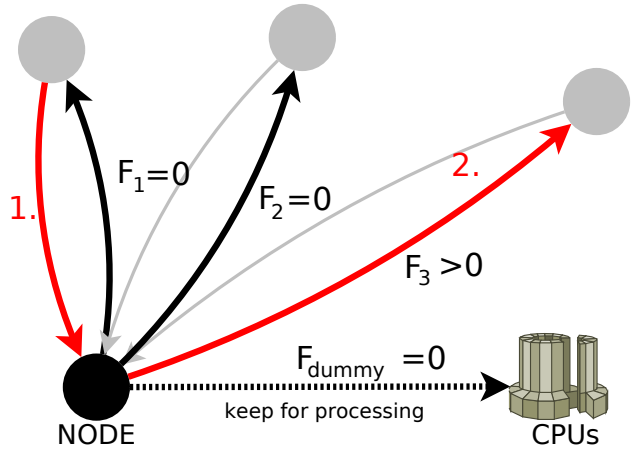
Solving procedure

1. Calculate capacities of dummy edges using monitoring data.
2. Solve the problem for output data flows.
3. Recalculate remaining network capacity.
4. Solve the problem for input transfers.

Plan execution

- A local handler at each node receives the plan:
  - Flows of outgoing edges - how much data of each type should be send over that link.
  - Flow of dummy edges - how much data should be processed at this node.
- When a new file arrives, the handler decides according to the plan and current state:
  - Process the file (transfer over dummy edge)
  - OR forward it over one of the links
- Decrease remaining flow of the link by the size of the file after transfer.





Enabling Caching

- Performance of cache algorithms implemented with watermarking concept was simulated for a wide scope of cache size and low marks. 3 access patterns of 2 different experiments were used as input for simulations. 27 algorithms were tested in 90 simulation setups.

Motivation

- Cache of data-transfer tools.
- Management of local data replicas.

Access patterns used for simulation

STAR1: RCF@BNL, **Tier-0** for STAR experiment, Xrootd log, user analysis, 3 months period (June-August 2012).

STAR2: RCF@BNL, **Tier-0** for STAR experiment, Xrootd log, user analysis, 7 months period (August 2012 - February 2013).

GOLIAS: FZU Prague, part of **Tier-2** of ATLAS. ATLAS and AUGER experiments, DPM log, user analysis + production, 3 months period (November 2012 - February 2013). AUGER makes less than 1% of total requests.

Selected caching algorithms

- **First-In-First-Out (FIFO)**: evicts files in the same order they entered the cache.
- **Least-Recently-Used (LRU)**: evicts the set of files which were not used for the longest period of time.
- **Least-Frequently-Used (LFU)**: evicts the set of files which were requested less times since they entered the cache.
- **Most Size (MS)**: evicts the set of files which have the largest size.
- **A**daptive **R**eplacement **C**ache (**ARC**). 2 lists: L1 - files with *access count* $= 1$, and L2 - files with *access count* > 1. LRU is applied to both list. The length of each list depends on $p = \textit{cache hits in } L1 / \textit{cache hits in } L2$.
- **L**east **V**alue based on **C**aching **T**ime (**LVCT**). Deletes files according to the value of the Utility Function.

$$UtilityFunction = \frac{1}{CachingTime \times FileSize} \tag{1}$$

where **Caching Time** of a file F is the sum of size of all files accessed after the last request for the file F.

What caching algorithm is the best?

Average improvement over FIFO

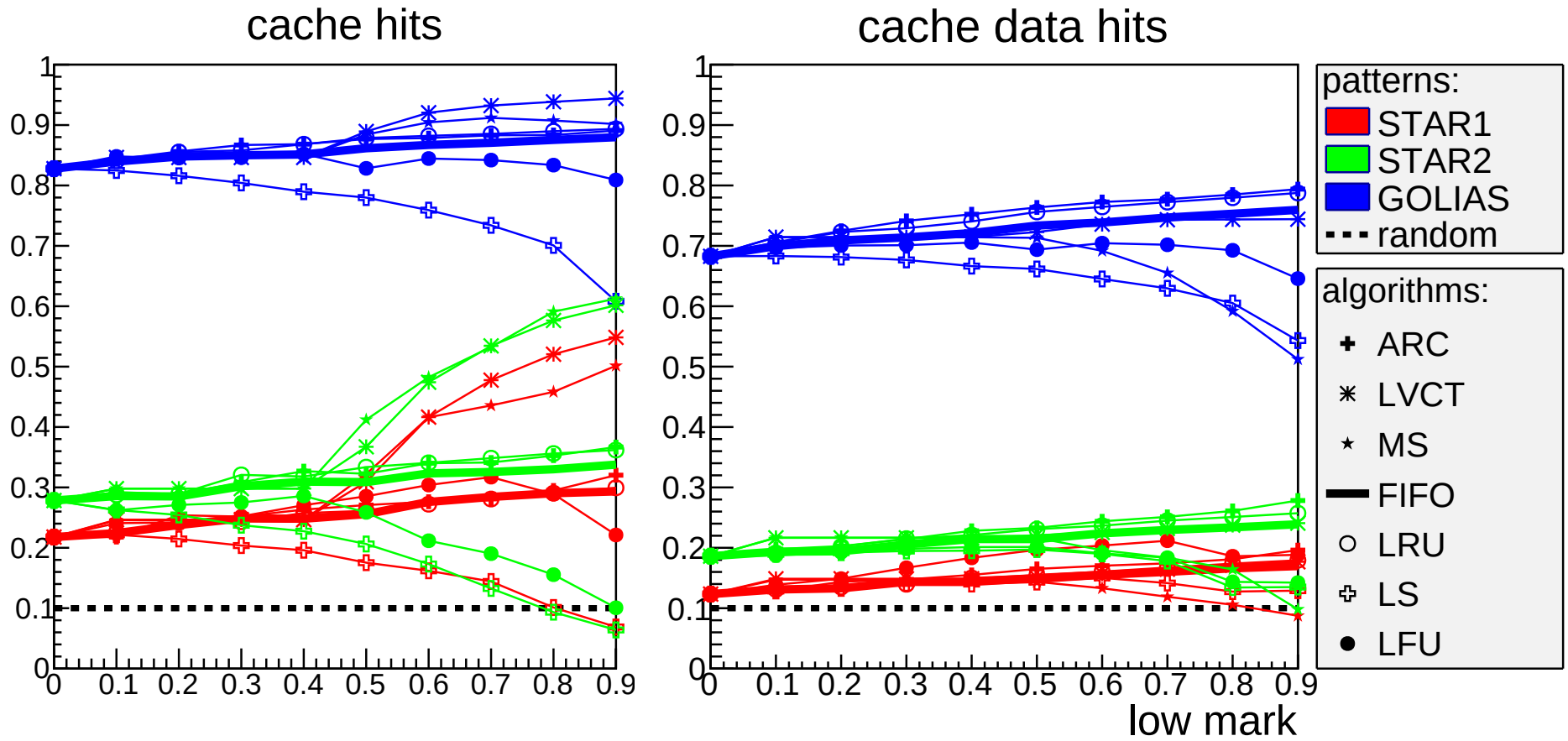
Algorithm	cache hits	cache data hits
MS	116 %	-20 %
LRU	8 %	5 %
LFU	-27 %	-18 %
ARC	13%	11%
LVCT	86 %	2 %

For studied access patterns

- Regardless of the cache size, Tier-level and specificity of experiment the LVCT and ARC appear to be the most efficient caching algorithms.
- If the goal is to minimize makespan due to a transfer startup overhead the LVCT algorithm should be selected.
- If the goal is to minimize the network load the ARC algorithm is an option.

Dependence of cache performance on low mark

cache size / storage size = 0.1 ,high mark = 0.95



- Difference between Tier-2 and Tier-0 leads to distinct cache performance.
- With higher low mark the number of clean-ups increases as well as overall performance.

Conclusion

- Automated and scalable planning and optimization of distributed computations are highly required in data intensive computational fields such as High Energy and Nuclear Physics.
- Resent works has revealed the potential of global planning for this task.
- A CSP for scheduling of data production over Grid was formulated.
- The simulations based on log data has shown that the proposed approach systematically provides a smaller makespan and adapts to the increase of transfer overheads better then the other simulated heuristics.
- Caching algorithms has been evaluated to meet the needs of distributed data production.
- Network flow model

Future plan

- Test the new approach using full scale Grid simulations with the help of modeling tools widely used in Grid research community (GridSim)
- Improve the planner performance in order to enable online scheduling in real environment.


M Zerola et al. "One click dataset transfer: toward efficient coupling of distributed storage resources and CPUs". In: *J. Phys.: Conf. Ser.* 368.012022 (2012).


Jan Balewski et al. "Offloading peak processing to virtual farm by STAR experiment at RHIC". In: *J. Phys.: Conf. Ser.* 368.012011 (2012).


Korea Institute of Science and Technology Information KISTI. http://en.kisti.re.kr/.


Olivier Beaumont et al. "Bandwidth-centric allocation of independent tasks on heterogeneous platforms". In: *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002*. IEEE. 2001, 6–pp.


Henri Casanova et al. "Heuristics for scheduling parameter sweep applications in grid environments". In: *Heterogeneous Computing Workshop, 2000.(HCW 2000) Proceedings. 9th*. IEEE. 2000, pp. 349–363.

 Elizeu Santos-Neto et al. "Exploiting replication and data reuse to efficiently schedule data-intensive applications". In: *Job Scheduling Strategies for Parallel Processing*. Springer. 2005, pp. 210–232.

 K Ranganathan and I Foster. "Decoupling computation and data scheduling in distributed data-intensive applications". In: *11th IEEE International Symposium on High Performance Distributed Computing* (2002), pp. 352–358.

 Pavel Troubil and Hana Rudová. "Integer linear programming models for media streams planning". In: *ICAOR* 11 (2011), pp. 509–522.