Workflow System for Data-intensive Many-task Computing

Masahiro Tanaka and Osamu Tatebe

University of Tsukuba, JST/CREST

Outline

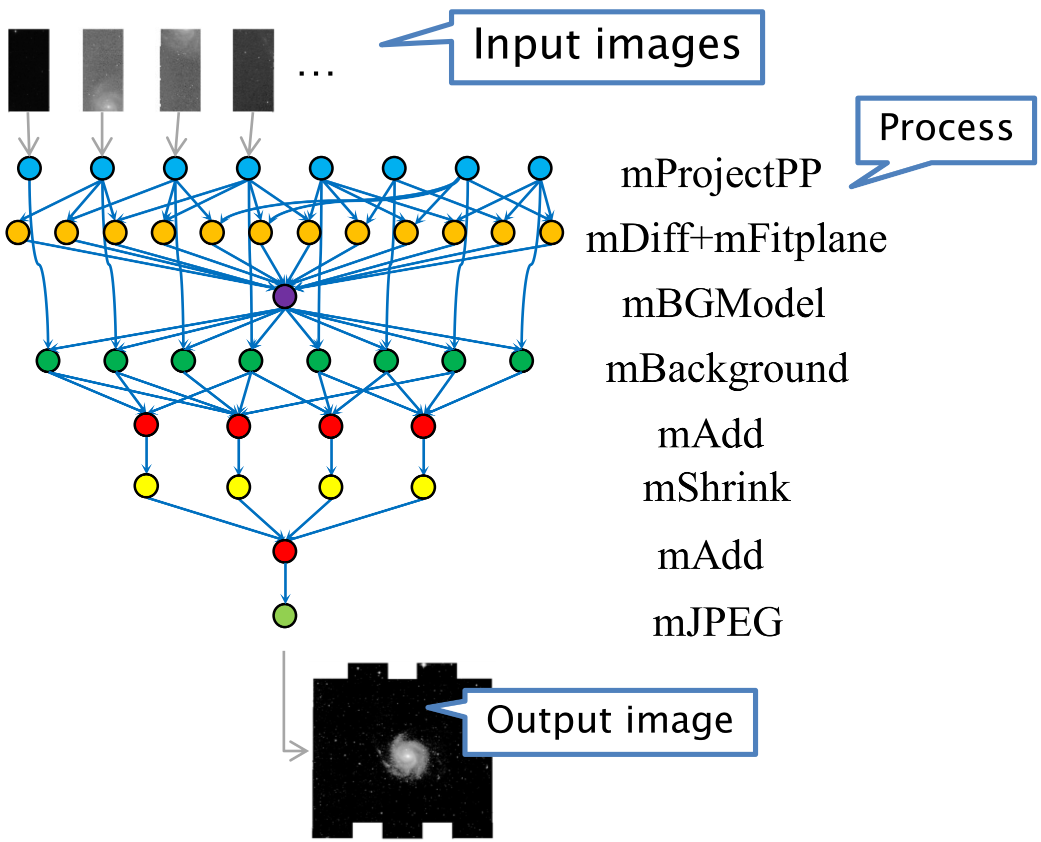
- Background
- Pwrake Workflow System
- IO-aware Task Scheduling
  - Locality-aware Task Scheduling using Multi-Constraint Graph Partitioning (MCGP)
  - Disk cache-aware Tasks Scheduling
- Conclusion

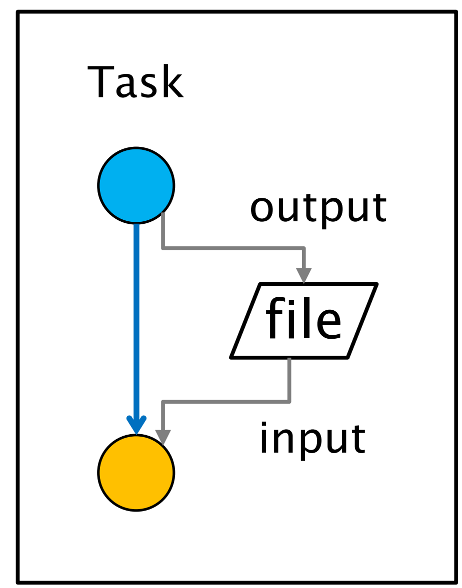
Background: Data-intensive Science

- Many Scientific Fields
  - Astronomy, Bioinformatics, Earth Science, Particle Physics, …
- Data I/O > Computation
  - Interaction through File System
- Handles huge amount of data
  - HSC for Subaru Telescope generates ~300GB/night.
  - Requires Parallel processing on Distributed Computer Systems

Example of scientific workflow: Montage (Astronomy image processing)

Workflow DAG





Pwrake Workflow System

Pwrake Workflow System

- Parallel Workflow extension to **Rake**
- Target: data-intensive and many-task scientific workflow
- **Pwrake** is based on:
  - **Rake** : Ruby version of UNIX Make
    - Workflow definition language for Many-Task Scientific Workflows
  - **Gfarm** : Distributed File System
    - Scalable I/O performance
    - Use local storage of compute nodes

Workflow Definition Language

- Task definition format
  - e.g. DAX
  - Need script to define many tasks.
- Design a new language
  - e.g. Swift (Wilde et al. 2011)
  - Learning cost, Niche community.
- Use an existing language
  - e.g. GXP Make (Taura et al. 2013)
  - Extension rule is not enough for scientific workflows.

Our solution: Rake – Ruby Make

- Build tool written in Ruby
- Widely-used tool in the Ruby community
- Rake is an internal DSL
  - Ruby is an host language
    - reduces learning cost
    - able to use Ruby language features

Useful features of Rake

- For-Loop

BASENAMES = Array of basenames

```
for i in BASENAMES
  file "out/#{i}.fits" => "src/#{i}.fits" do |t|
    sh "mProjectPP #{t.prerequisites[0]} #{t.name} region.hdr"
  end
end
```

- Complex Rule using script

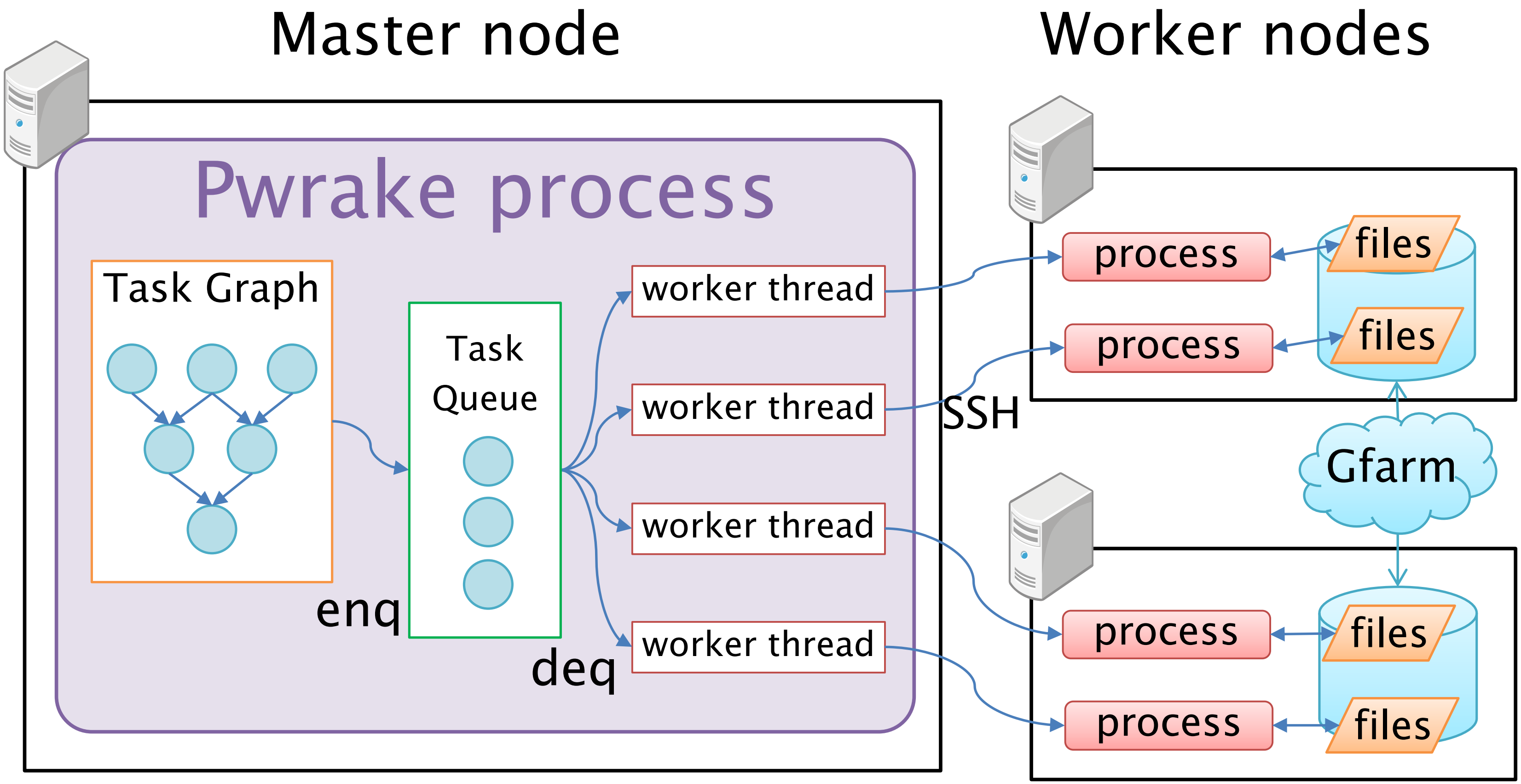
FILEMAP = Mapping input files to output files

```
rule /^d¥/.*¥.fits$/ => proc{|x| FILEMAP[x]} do |t|
  p1,p2 = t.prerequisites
  sh "mDiff #{p1} #{p2} #{t.name} region.hdr"
end
```

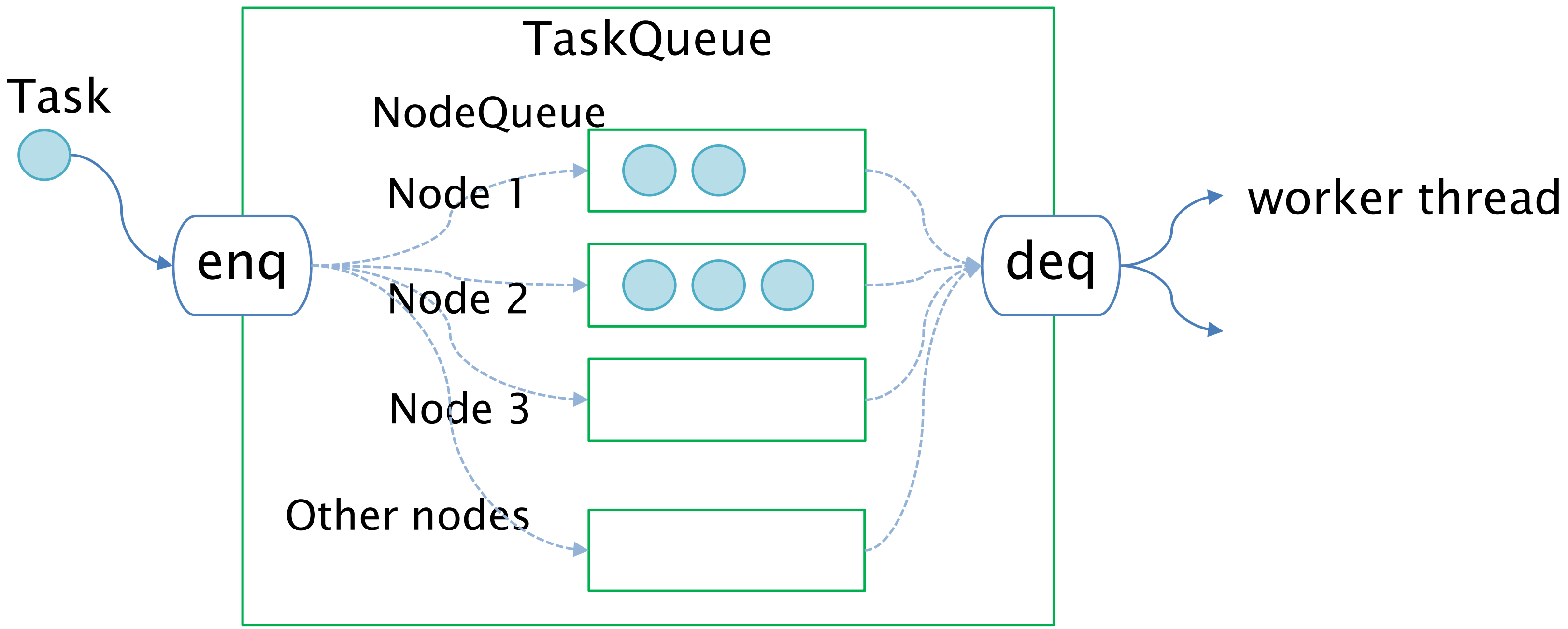
Design of Pwrake

- Inherit Rake
  - Workflow (task) definition language
  - File-based task dependency
    - Resume/Restart
  - Implementation, e.g., Task class, Application module
- Implement Pwrake extension
  - remote process execution
  - parallel task execution
  - task queue (includes scheduling)
  - find file location using Gfarm API

Pwrake Archetecture

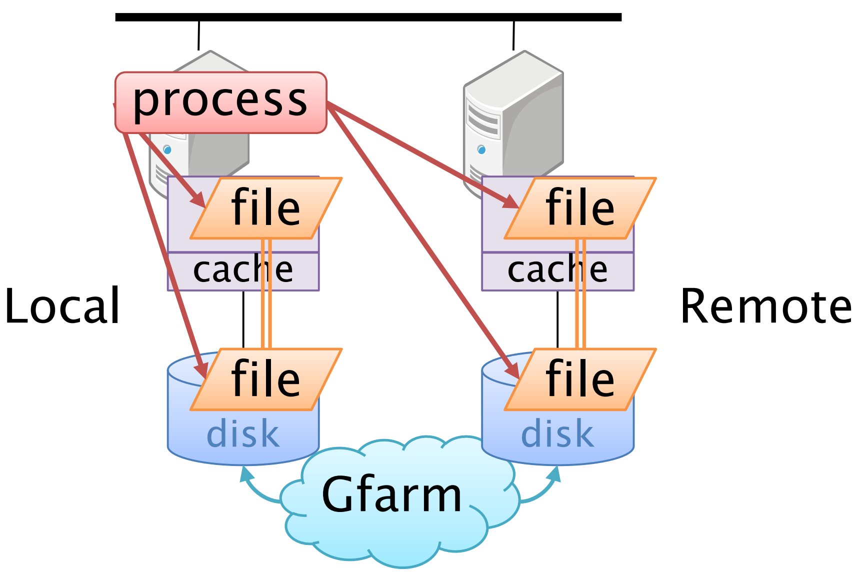


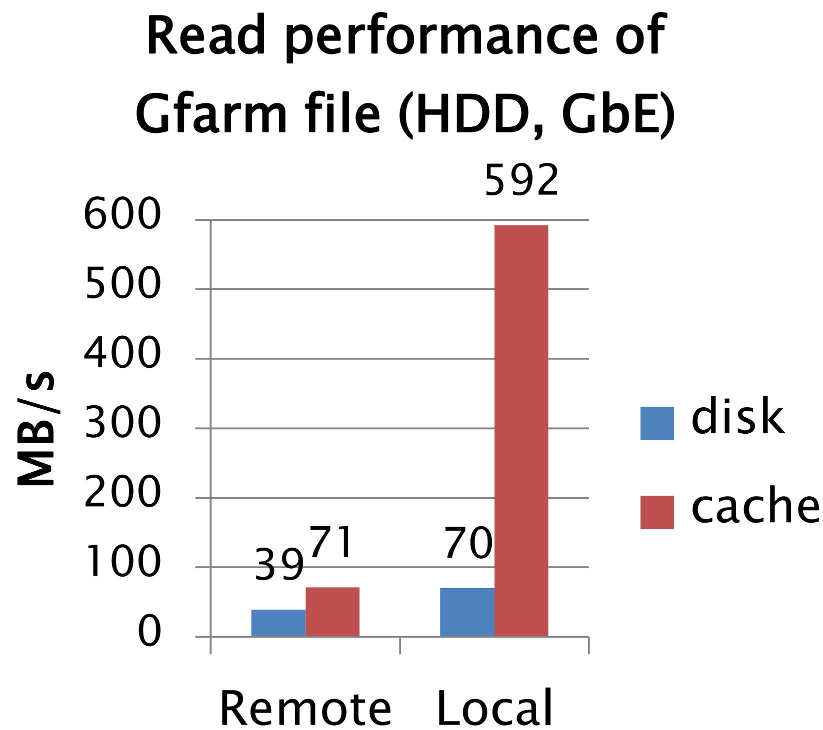
Design of Task Queue



I/O-aware Task Scheduling

- Issues:
  - File Locality
  - Disk cache
    - (buffer/page cache)





Locality-aware Scheduling based on MCGP

(Multi-Constraint Graph Partitioning)

(CCGrid 2012)

Locality-aware Scheduling Methods

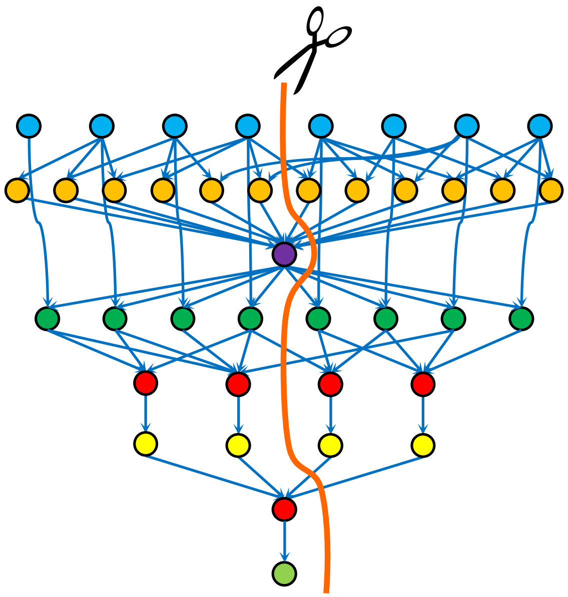
1. Naïve locality scheduling
   - Assign a task to a node where its input file is stored.

2. Method using MCGP (Multi-Constraint Graph Partitioning)
   - Our proposal (CCGrid 2012)

(Idle workers steal tasks)

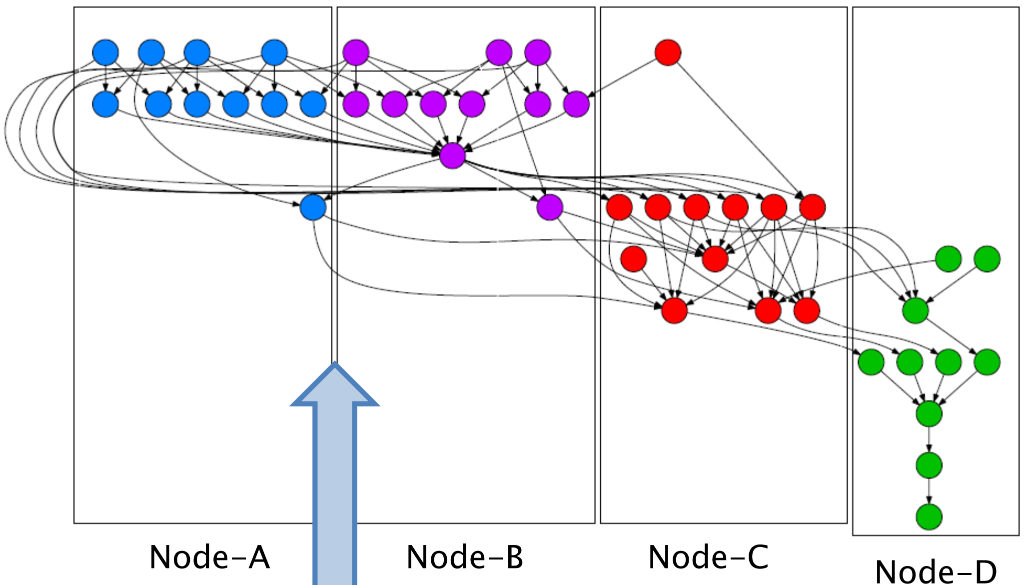
Graph Partitioning ⇔ Task Scheduling

- Is Graph Partitioning also applicable to Workflow DAG?
  - Vertex ⇔ Computation
  - Edge ⇔ Communication
  - Minimize:
    - Edge-cut ⇔ Data movement

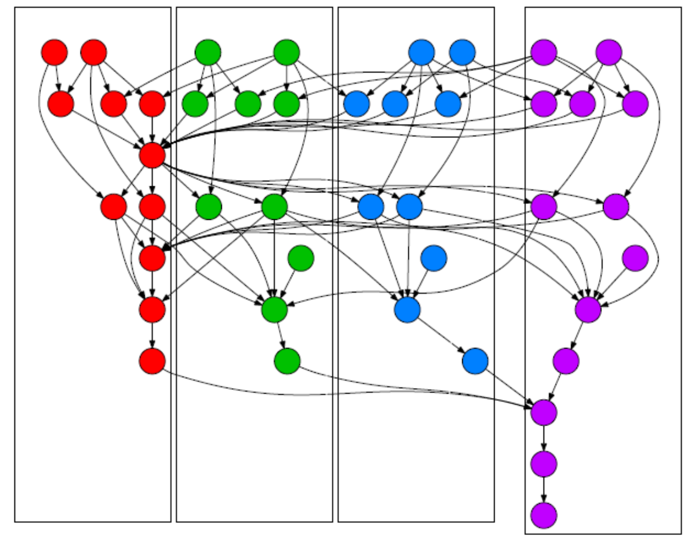


Graph Partitioning on DAG

Standard Graph Partitioning



Ideal Partitioning for Scheduling



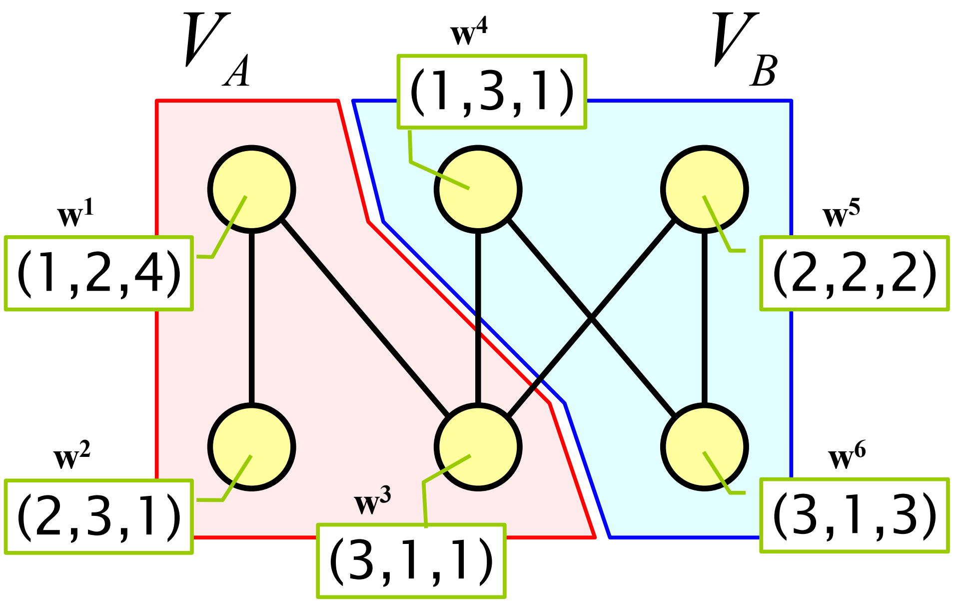
Former Tasks

Latter Tasks

Standard GP is not aware of parallelizable tasks

Multi-Constraint Graph Partitioning (MCGP)

Vertex Weight Vectors $\mathbf{w}^i = \left(w_1^i, w_2^i, w_3^i\right)$



V_A V_B

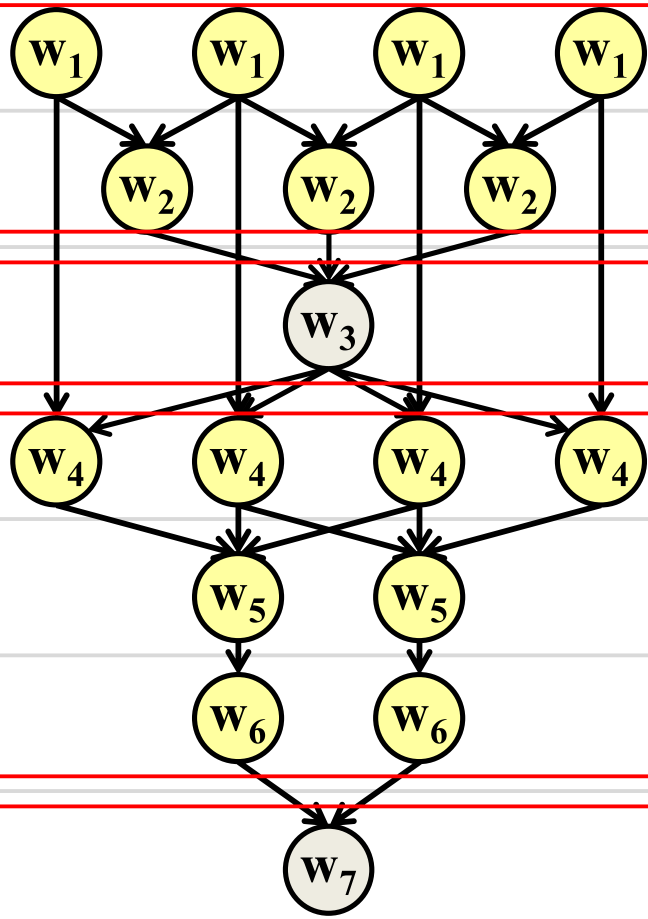
1st dim: $\sum_{i \in V_A} w_1^i = \sum_{j \in V_B} w_1^j = 6$

2nd dim: $\sum_{i \in V_A} w_2^i = \sum_{j \in V_B} w_2^j = 6$

3rd dim: $\sum_{i \in V_A} w_3^i = \sum_{j \in V_B} w_3^j = 6$

Balance the sum of Vertex Weights at each dimension

Proposed method: MCGP
(Multi-Constraint Graph Partitioning)



$w_1 = (1,0,0,0,0)$

$w_2 = (0,1,0,0,0)$

$w_3 = (0,0,0,0,0)$

$w_4 = (0,0,1,0,0)$

$w_5 = (0,0,0,1,0)$

$w_6 = (0,0,0,0,1)$

$w_7 = (0,0,0,0,0)$

$N_{task} \geqq N_{group}$:
- set 1 at i^{th} dim
- set 0 at others

$N_{task} < N_{group}$:
- set 0 to all dims

Result of MCGP

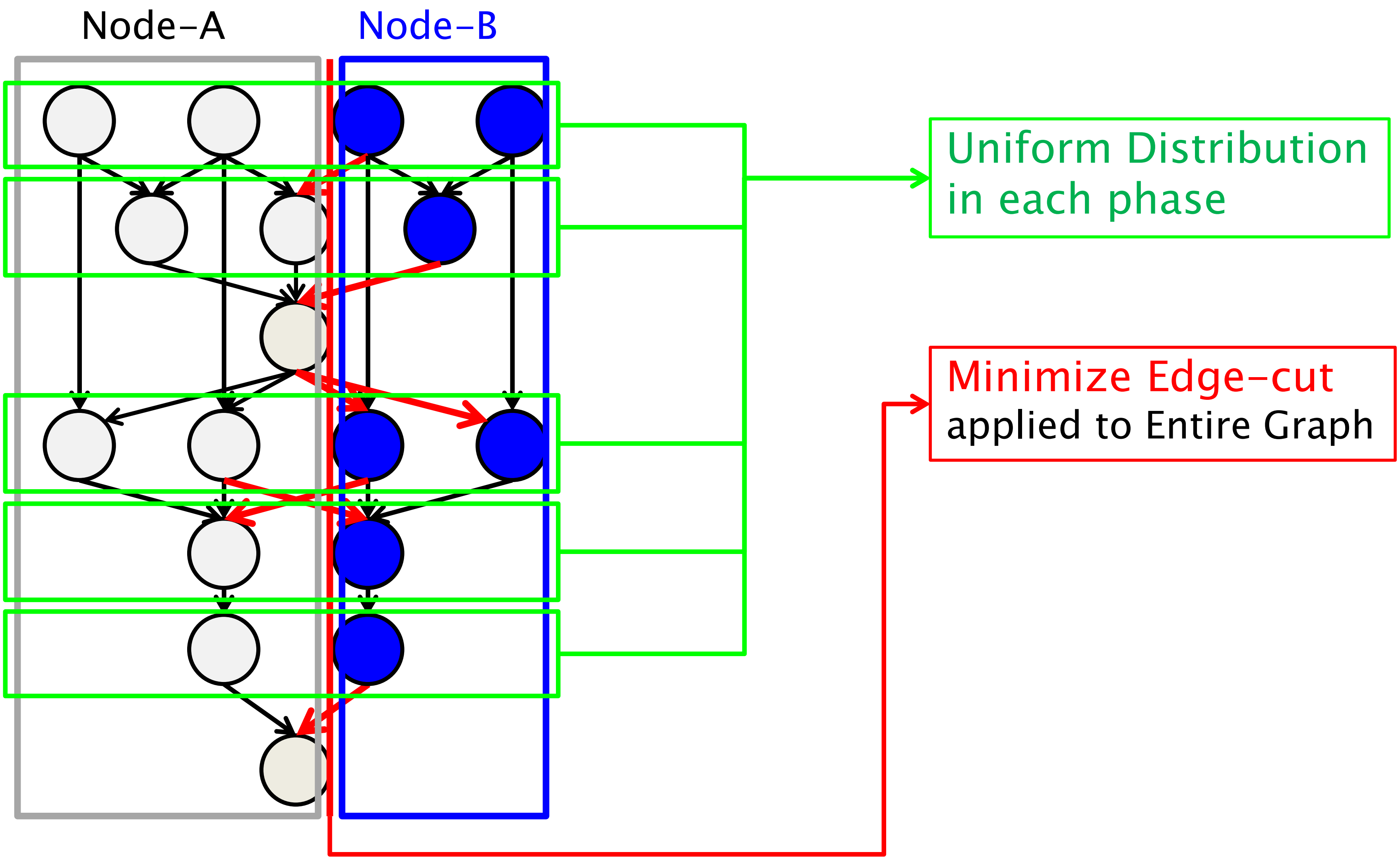
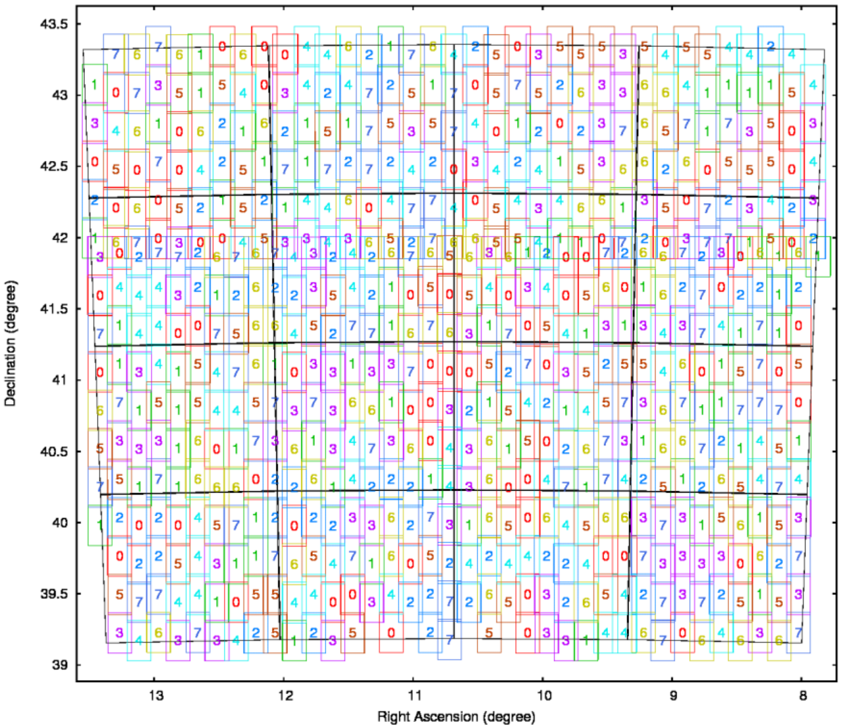


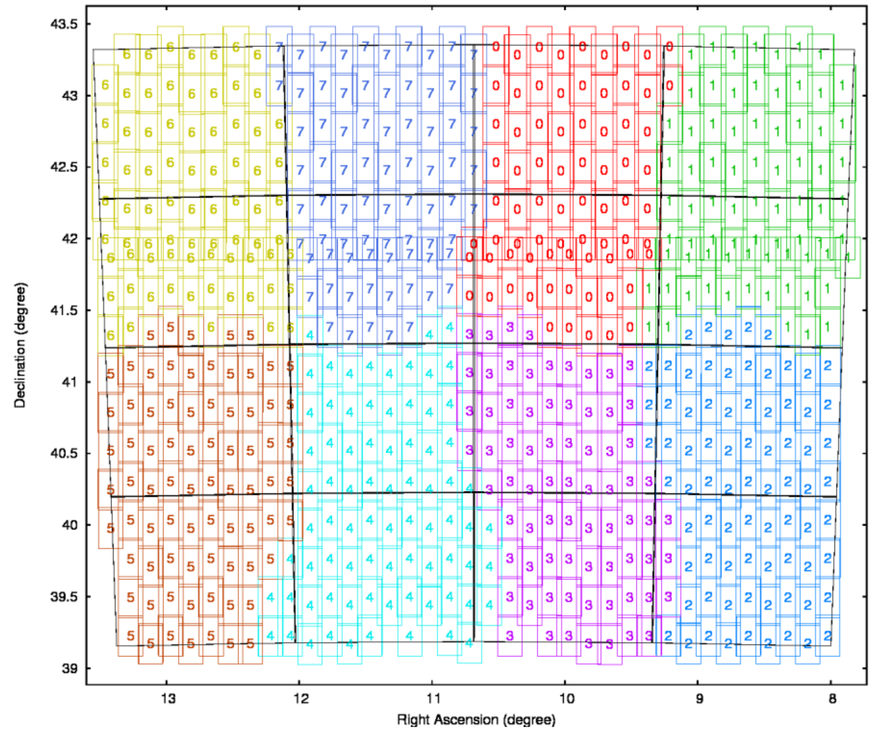
Image Position and Task Nodes

(Initially, input files are stored in one node)

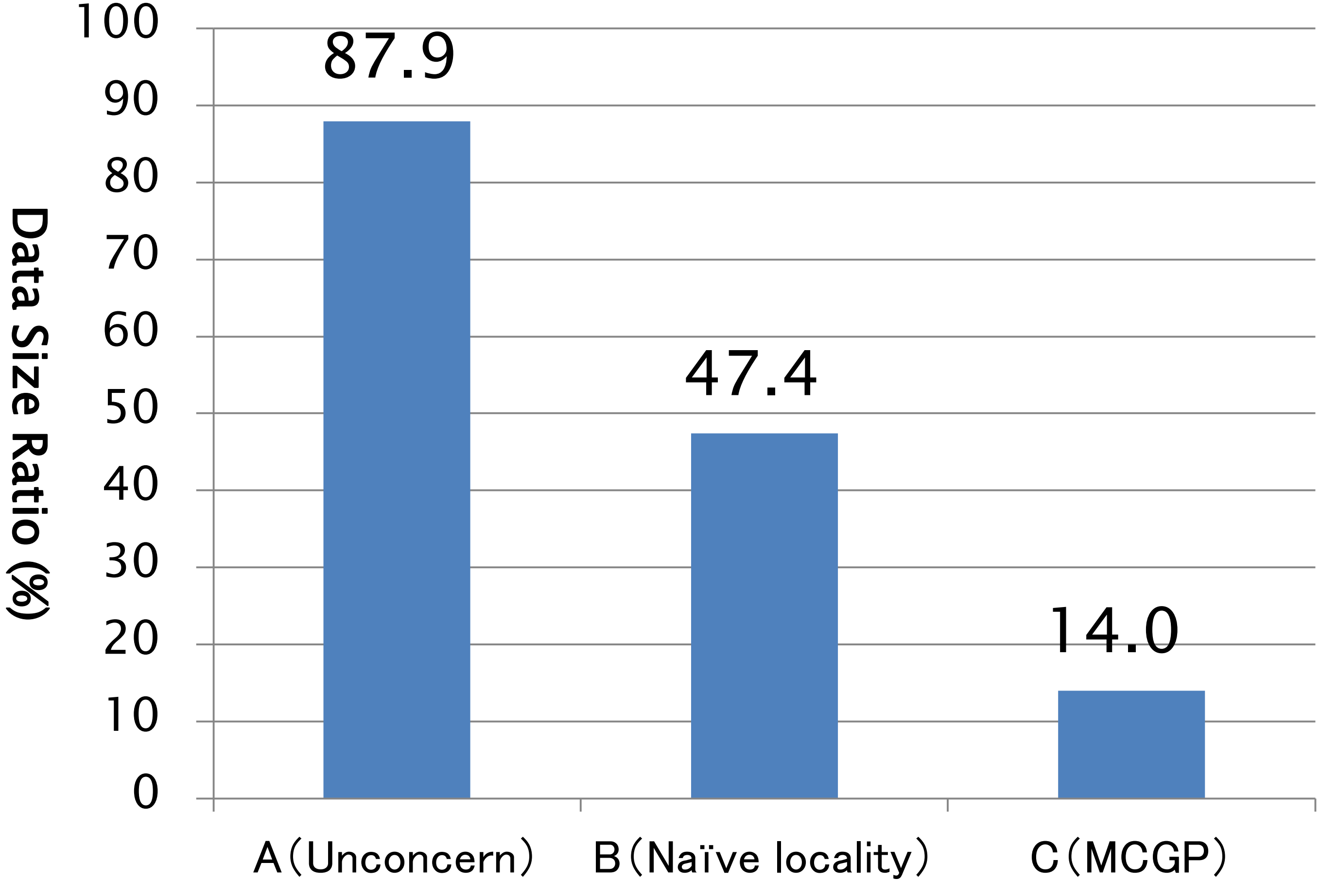
Naïve locality



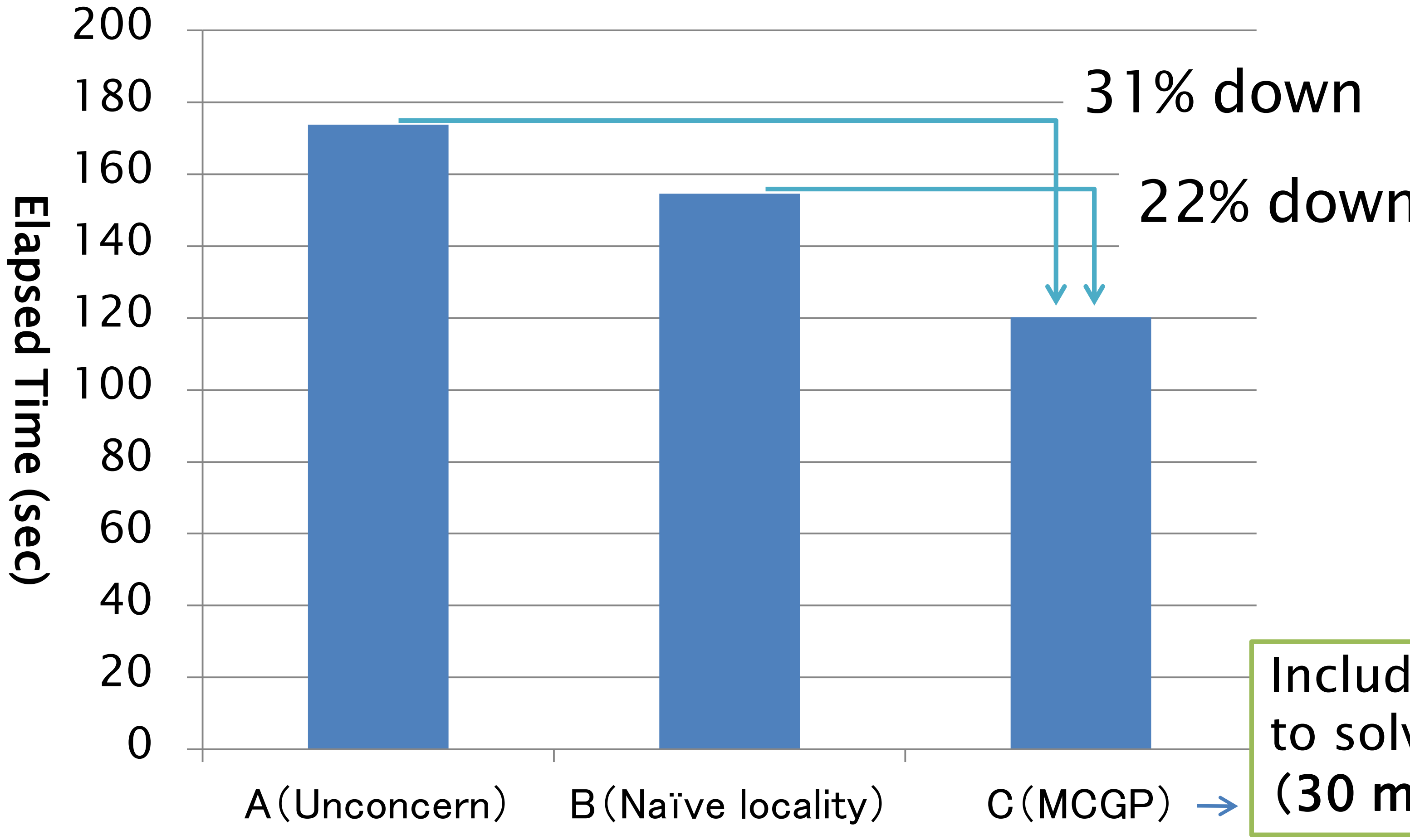
MCGP



Data Movement between nodes



Workflow Execution Time



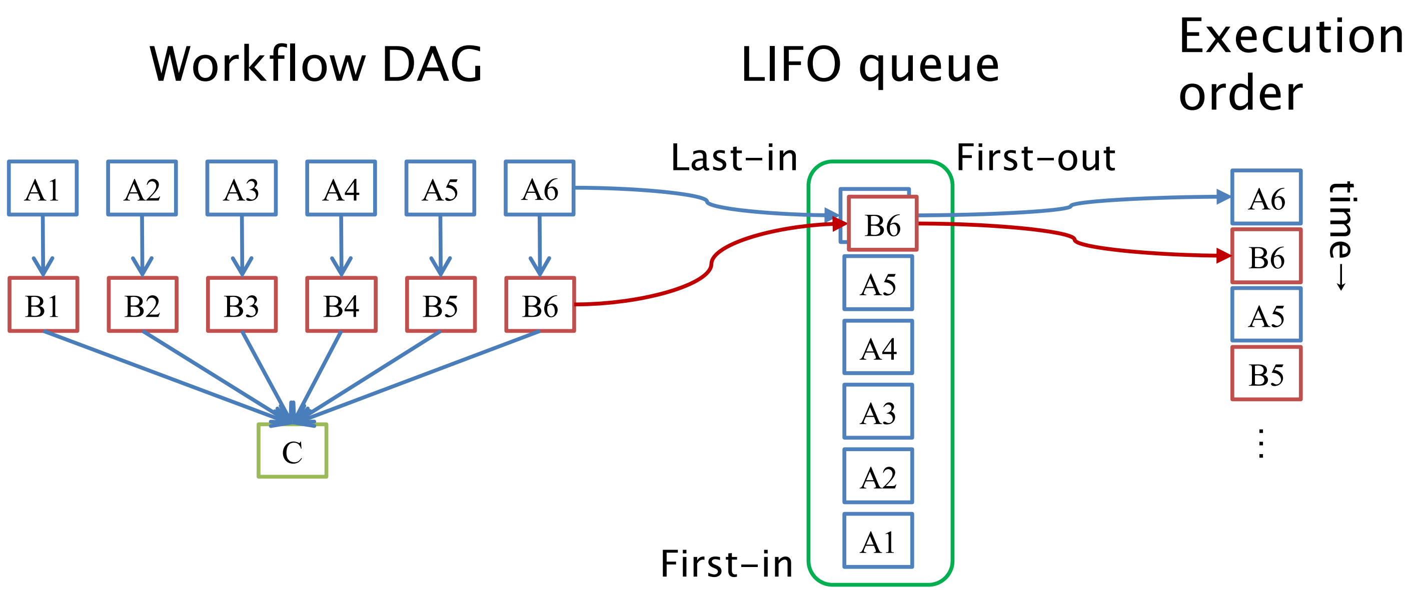
Includes time to solve MCGP **(30 ms)**

Disk cache-aware Task Scheduling

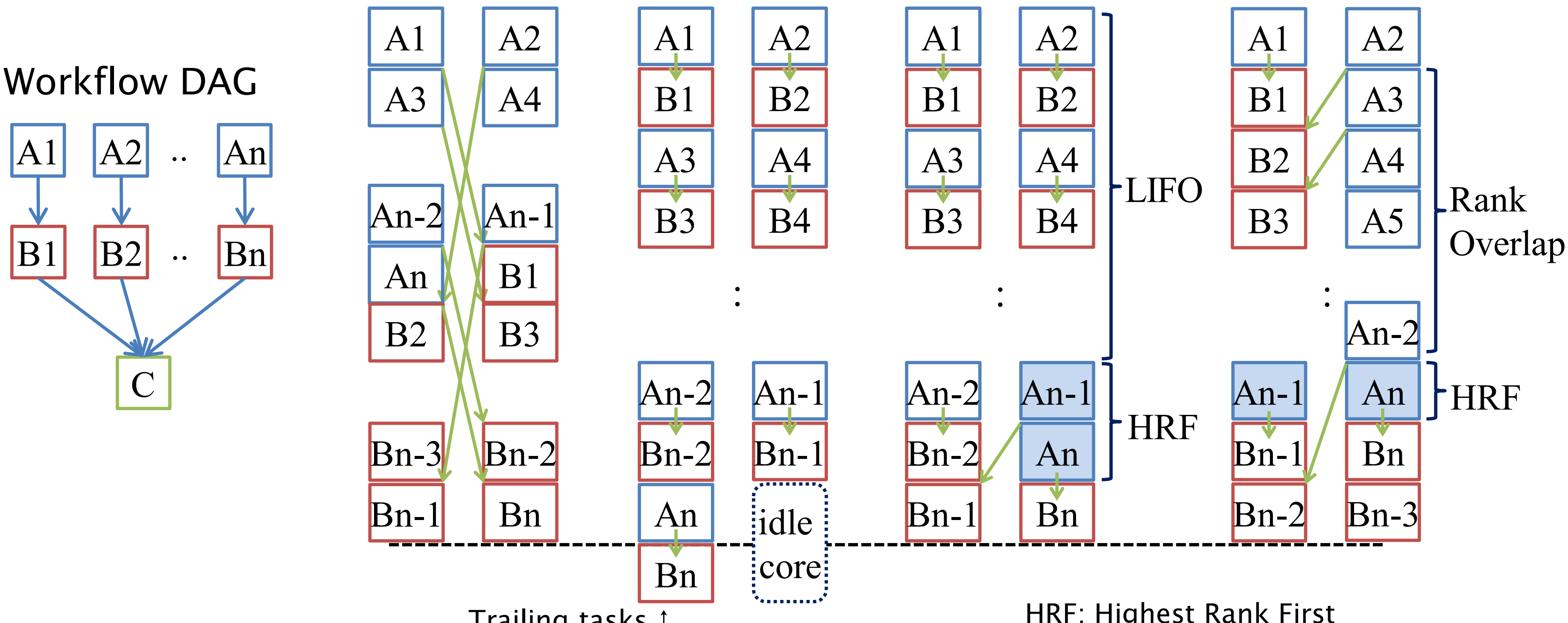
(Cluster 2014)

Disk Cache-aware Task Scheduling

- **LIFO** queue:
  - Intermediate files are read soon.
  - High probability that the file is cached.
  - **Trailing task problem** (Armstrong et al. MTAGS 2010)



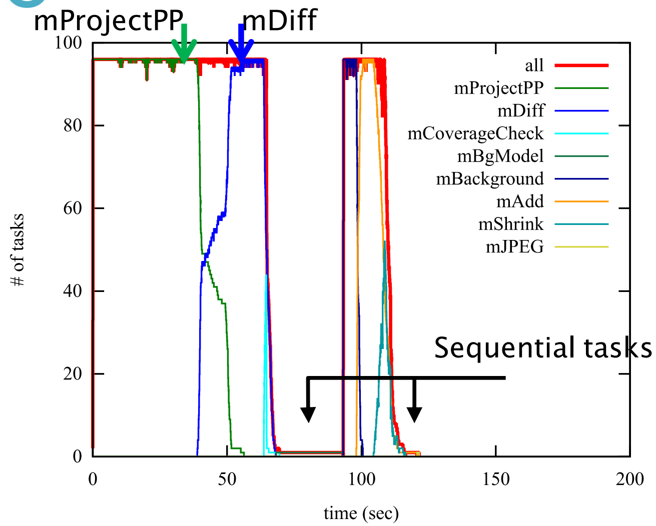
Proposed Scheduling methods



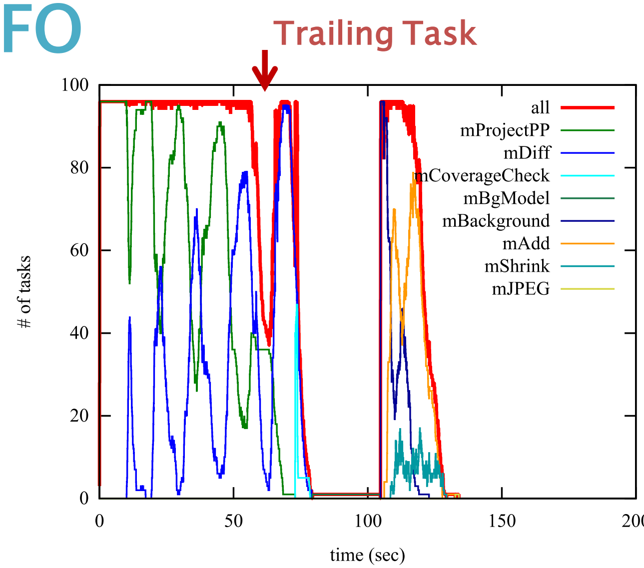
	FIFO (HRF)	LIFO	LIFO+HRF	Rank Equalization+HRF
Disk Cache	×	◎	◎	○
Trailing Task	○	×	○	○
Task Overlap	×	×	×	○

Core Utilization

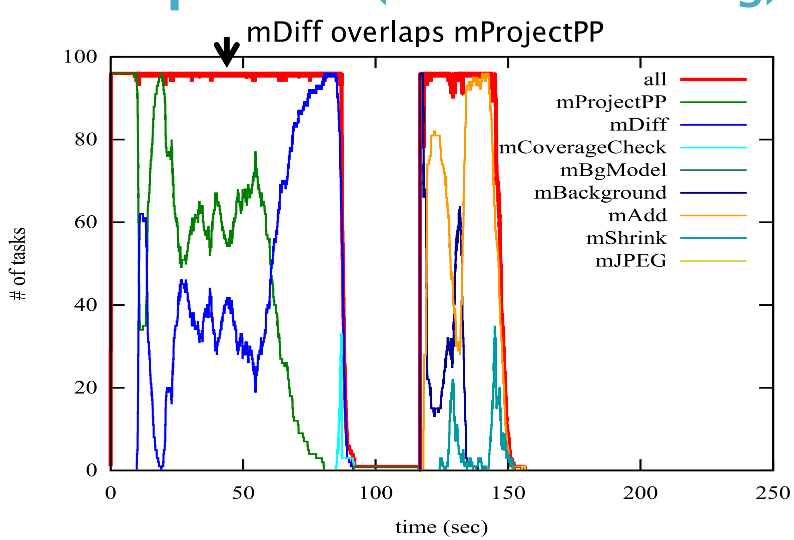
FIFO



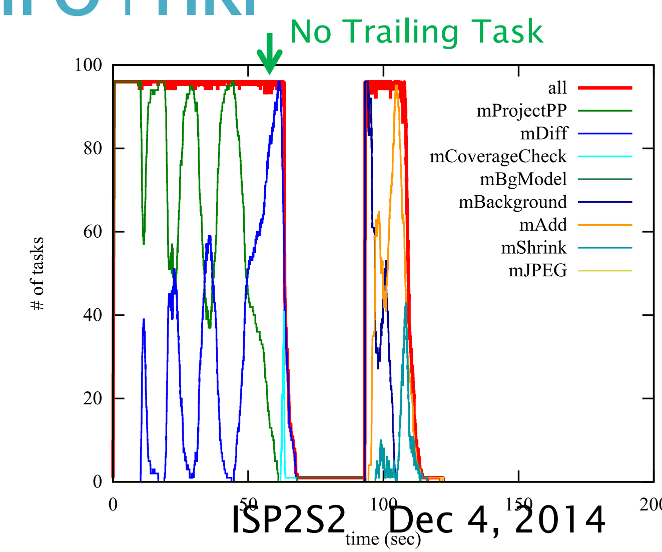
LIFO



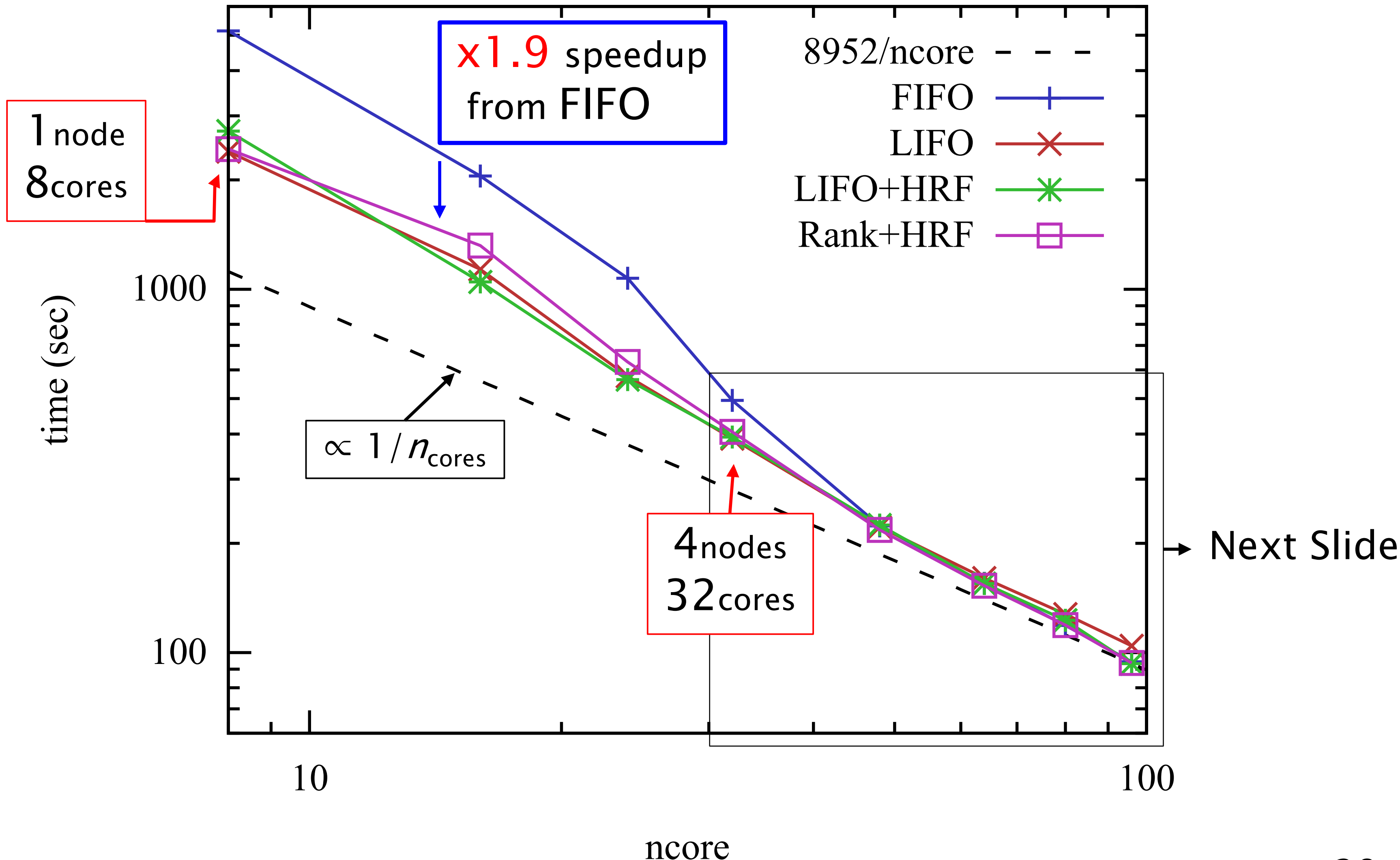
Rank Eq+HRF (different setting)



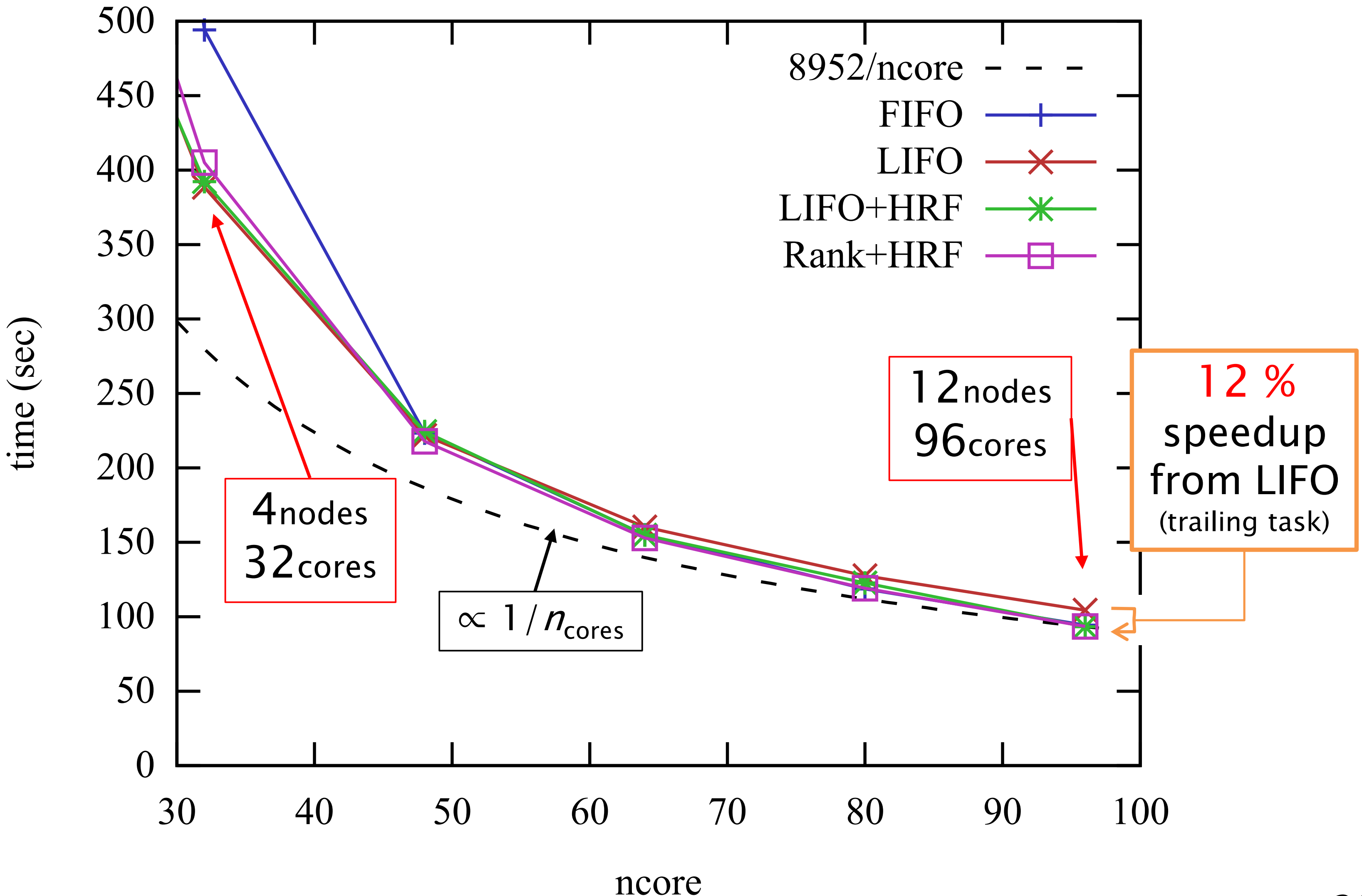
LIFO+HRF



Measurement of Strong Scaling
(1–12 nodes, Logarithmic)



Measurement of Strong Scaling
(4–12 nodes, Linear)



Conclusion

- We developed Pwrake workflow system for data-intensive, many-task workflows.
- I/O-aware workflow scheduling:
  - Locality-aware scheduling using MCGP
    - remote file access: 88% ⇒ 14%
    - workflow execution: 31% speedup
  - Disk cache-aware scheduling
    - LIFO: 1.9x speedup
    - HRF: ~12% speedup