A Survey of Distributed Task Schedulers

Kei Takahashi (M1)

What do you want to do on a grid?

- Vast computing resources
  - Calculation power
  - Memory
  - Data storage
- Large scale computation
  - Numerical simulations
  - Statistical analyses
  - Data mining

.. for everyone

Grid Applications

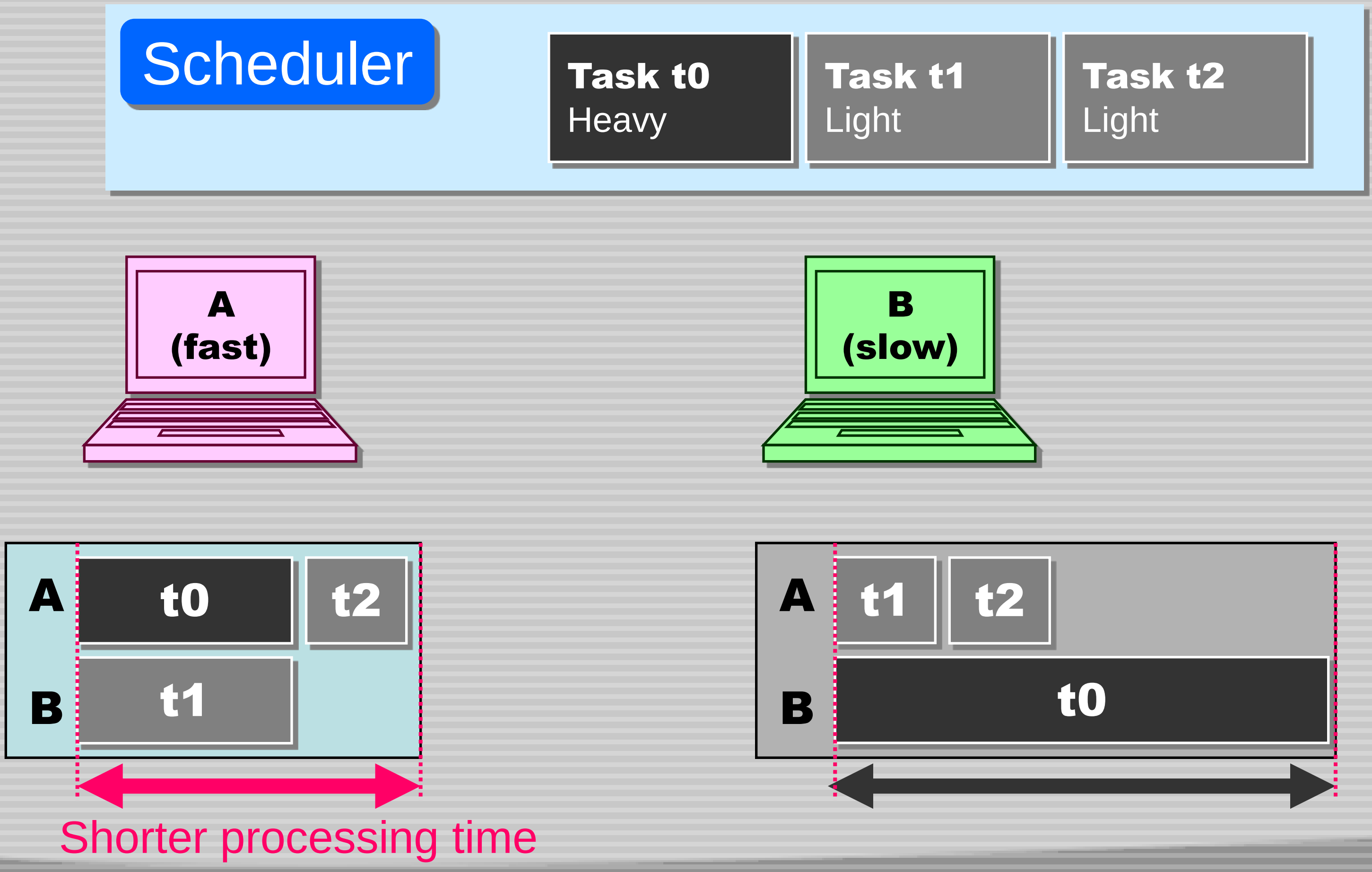
- For some applications, it is inevitable to develop parallel algorithms
  - Dedicated to parallel environment
  - E.g. matrix computations
- However, many applications are efficiently sped up by simply running multiple serial programs in parallel
  - E.g. many data intensive applications

Grid Schedulers

- A system which distributes many serial tasks onto the grid environment
  - Task assignments
  - File transfers
- A user need not rewrite serial programs to execute them in parallel
- Some constraints need to be considered
  - Machine availability
  - Machine spec (CPU/Memory/HDD), load
  - Data location
  - Task priority

An Example of Scheduling

- Each task is assigned to a machine



Efficient Scheduling

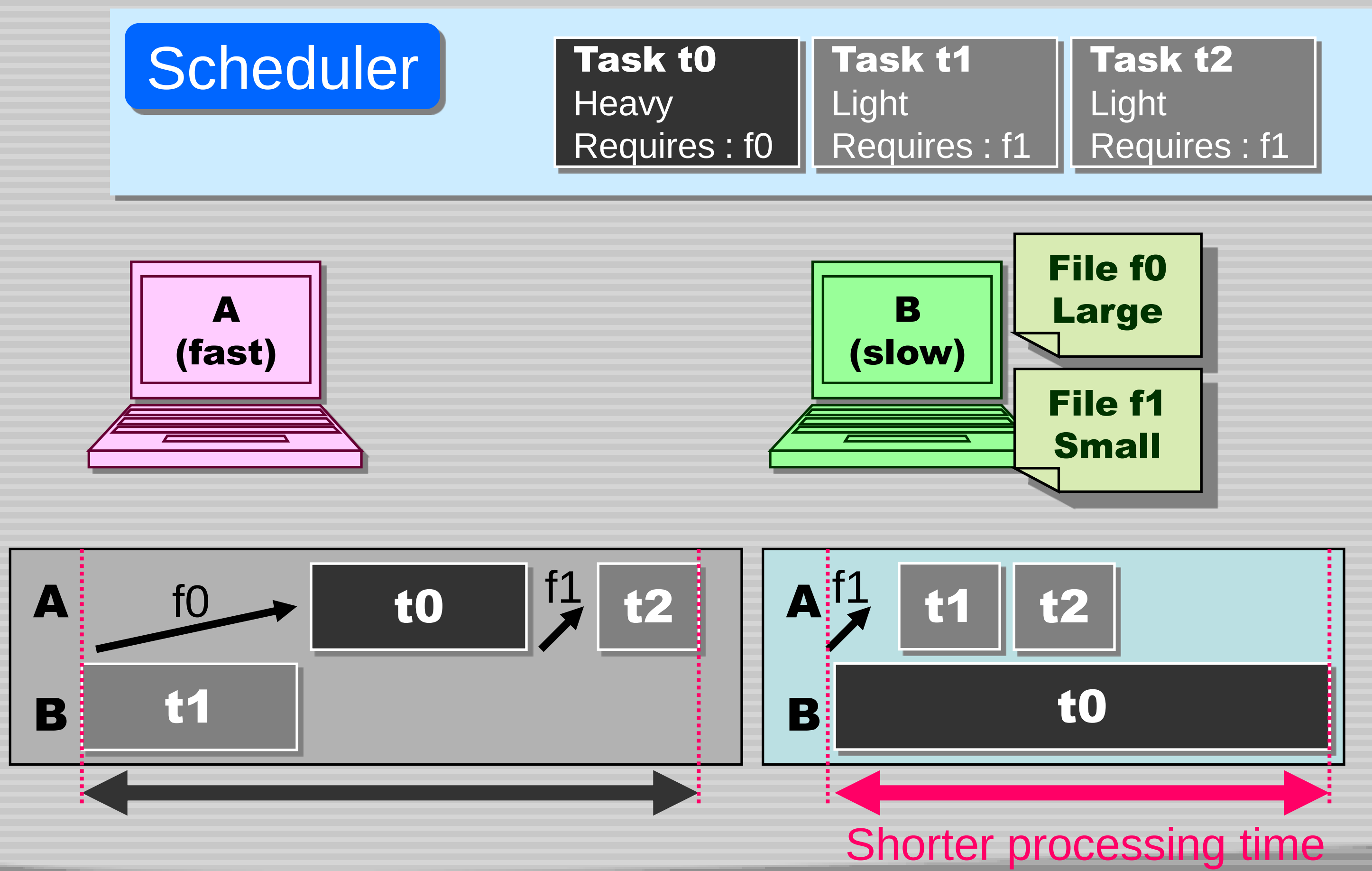
- Task scheduling in heterogeneous environment is not a new problem. Some heuristics are already proposed.
- However, existing algorithms could not appropriately handle some situations
  - Data intensive applications
  - Workflows

Data Intensive Applications

- A computation using large data
  - Some gigabytes to petabytes
- A scheduler need to consider the followings:
  - File transfer need to be diminished
  - Data replica should be effectively placed
  - Unused intermediate files should be cleared

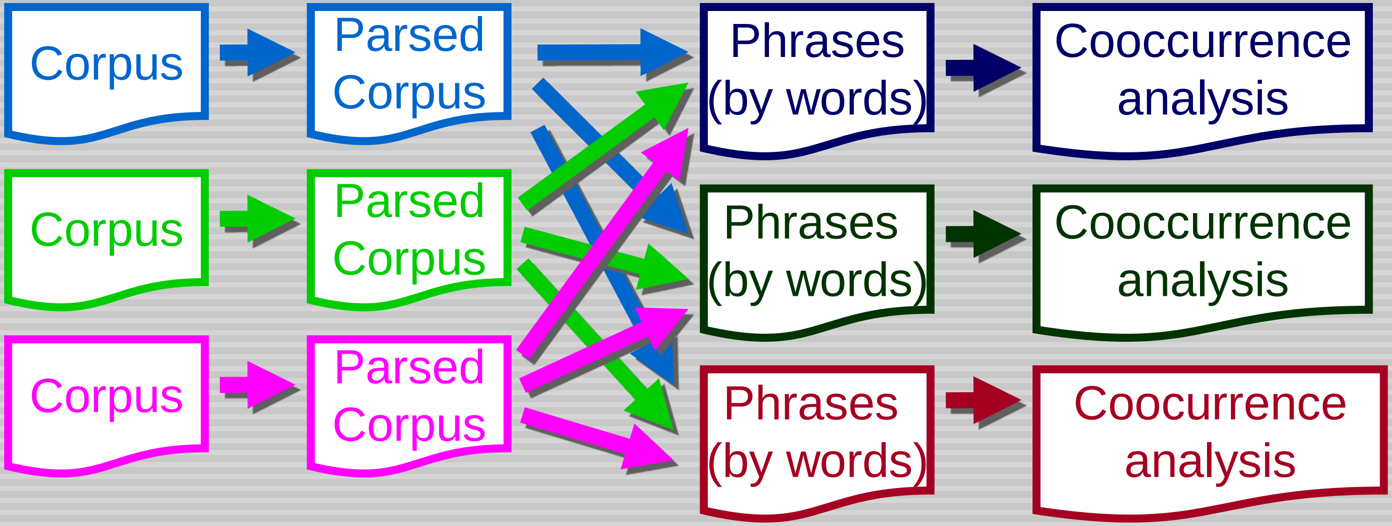
An Example of Scheduling

- Each task is assigned to a machine



Workflow

- A set of tasks with dependencies
  - Data dependency between some tasks
  - Expressed by a DAG



Workflow (cont.)

- Workflow is suitable for expressing some grid applications
  - Only necessary dependency is described by a workflow
  - A scheduler can adaptively map tasks to the real node environment
- More factors to consider
  - Some tasks are important to shorten the overall makespan

Agenda

- Introduction
- Basic Scheduling Algorithms
  - Some heuristics
- Data-intensive/Workflow Schedulers
- Conclusion

Basic Scheduling Heuristics

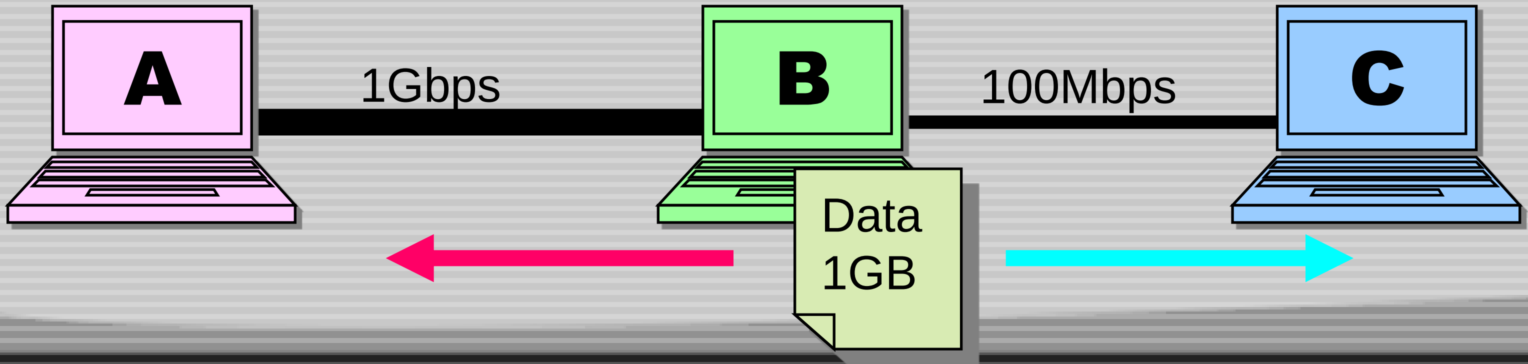
- Given information :
  - *ETC* (expected completion time) for each pair of a node and a task, including data transfer cost
  - No congestion is assumed
- Aim : minimizing the *makespan* (Total processing time)

[1] Tracy et al. A Comparison Study of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems (TR-ECE 00-04)

An example of ETC

- ETC of (task, node)
  = (node available time)
  + (data transfer time)
  + (task process time)

	Available after	Transfer	Process	ETC
Node A	200 (sec)	10 (sec)	100 (sec)	310 (sec)
Node B	0 (sec)	0 (sec)	100 (sec)	100 (sec)
Node C	0 (sec)	100 (sec)	20 (sec)	120 (sec)



Scheduling algorithms

- An ETC matrix is given
  - When a task is assigned to a node, the ETC matrix is updated
- An ETC matrix is consistent { if node M0 can process a task faster than M1, M0 can process every other task faster than M }
  - The makespan of an inconsistent ETC matrix differs more than that of a consistent ETC matrix

	Assigned to A Task 0	Task 1	Task 2
Node A	8	~~6~~ **14**	~~2~~ **10**
Node B	1	9	3
Node C	5	8	4

Greedy approaches

- Principles
  - Assign a task to the best node at a time
  - Need to decide the order of tasks
- Scheduling priority
  - Min-min : Light task
  - Max-min : Heavy task
  - Sufferage : A task whose completion time differs most depending on the node

Max-min / Min-min

- Calculate completion times for each task and node
- For each task take the minimum completion time
- Take one from unscheduled tasks
  - Min-min : Choose a task which has “max” value
  - Max-min : Choose a task which has “max” value
- Schedule the task to the best node

	Task 0	Task 1	Task 2
node A	8	6 **Max-min**	2
node B	1 **Min-min**	9	3
node C	5	8	4

Sufferage

- For each task, calculate *Sufferage* (The difference between the minimum and second minimum completion times)
- Take a task which has maximum Sufferage
- Schedule the task to the best node

	Task 0	Task 1	Task 2
Node A	8	6	2 **Sufferage = 1**
Node B	1 **Sufferage = 4**	9 **Sufferage = 2**	3
Node C	5	8	4

Comparing Scheduling Heuristics

- A simulation was done to compare some scheduling tactics [1]
  - Greedy (Max-min / Min-min)
  - GA, Simulated annealing, A*, etc.
- ETC matrices were randomly generated
  - 512 tasks, 8 nodes
  - Consistent, inconsistent
- GA performed the shortest makespan in most cases, however the calculation cost was not negligible
- Min-min heuristics performed well (at most 10% worse than the best)

[1] Tracy et al. A Comparison Study of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems (TR-ECE 00-04)

(Agenda)

- Introduction
- Scheduling Algorithms
- Data-intensive/Workflow Schedulers
  - GrADS
  - Phan’s approach
- Conclusion

Scheduling Workflows

- Additional Conditions to be considered
  - Task dependency
    - Every required file need to be transferred to the node before the task starts
    - "Non-executable" schedule exists
  - Data are dynamically generated
    - The file location is not known in advance
    - Intermediate files are not needed at last

GrADS [1]

- Execution time estimation
  - Profile the application behavior
    - CPU/memory usage
    - Data transfer cost
- Greedy scheduling heuristics
  - Create ETC matrix for assignable tasks
  - After assigning a task, some tasks turn to “assignable”
  - Choose the best schedule from Max-min, min-min and Sufferage

[1] Mandal. et al. "Scheduling Strategies for Mapping Application Workflows onto the Grid“ in IEEEInternational Symposium on High Performance Distributed Computing (HPDC 2005)

GrADS (cont.)

- An experiment was done on real tasks
  - The original data (2GB) was replicated to every cluster in advance
  - File transfer occurs in clusters
- Comparing to random scheduler, it achieved 1.5 to 2.2 times better makespan

Scheduling Data-intensive Applications [1]

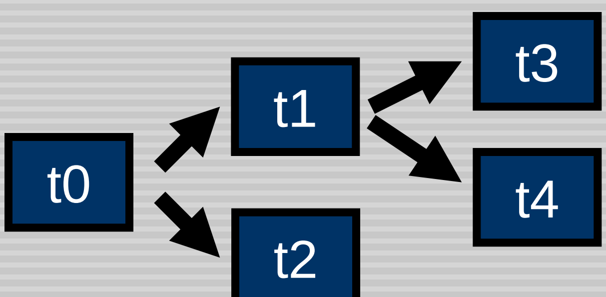
- Co-scheduling tasks and data replication
- Using GA
  - A gene contains the followings:
    - Task order in the global schedule
    - Assignment of tasks to nodes
    - Assignment of replicas to nodes
  - Only part of the tasks are scheduled at a time
    - Otherwise GA takes too long time

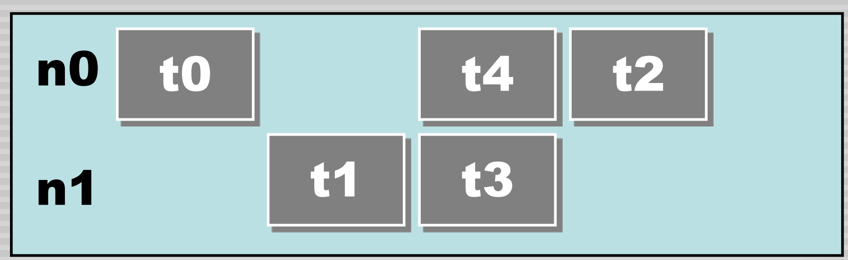
[1] Phan et al. “Evolving toward the perfect schedule: Co-scheduling task assignments and data replication in wide-area systems using a genetic algorithm.” In *Proceedings of the11th Workshop on task Scheduling Strategies for Parallel Processing*. Cambridge, MA. Springer-Verlag, Berlin, Germany.

(cont.)

- An example of the gene
  - One schedule is expressed in the gene

Task order	t0	t1	t4	t3	t2
Task assignment	t0:n0	t1:n1	t2:n0	t3:n1	t4:n0
Replicas	d0:n0	d1:n1	d2:n0		





(cont.)

- A simulation was performed
  - Compared to min-min heuristics with randomly distributed replicas
  - Number of GA generations are fixed (100)
- When 40 tasks are scheduled at a time, GA performs twice better makespan
- However, the difference decreases when more tasks are scheduled at a time
  - GA has not reached the best solution

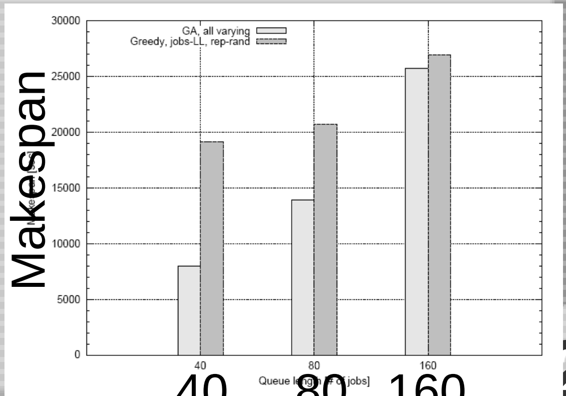


Fig. 9. Makespans for different queue lengths.

Conclusion

- Some scheduling heuristics were introduced
  - Greedy (Min-min, Max-min, Sufferage)
- GrADS can schedule workflows by predicting node performance and using greedy heuristics
- A research was done to use GA and co-schedule tasks and data replication

Future Work

- Most of the research is still on simulation
  - Hard to predict program/network behavior
- A scheduler will be implemented
  - Using network topology information
  - Managing Intermediate files
  - Easy to install and execute