Metode şi Algoritmi de Planificare (MAP)

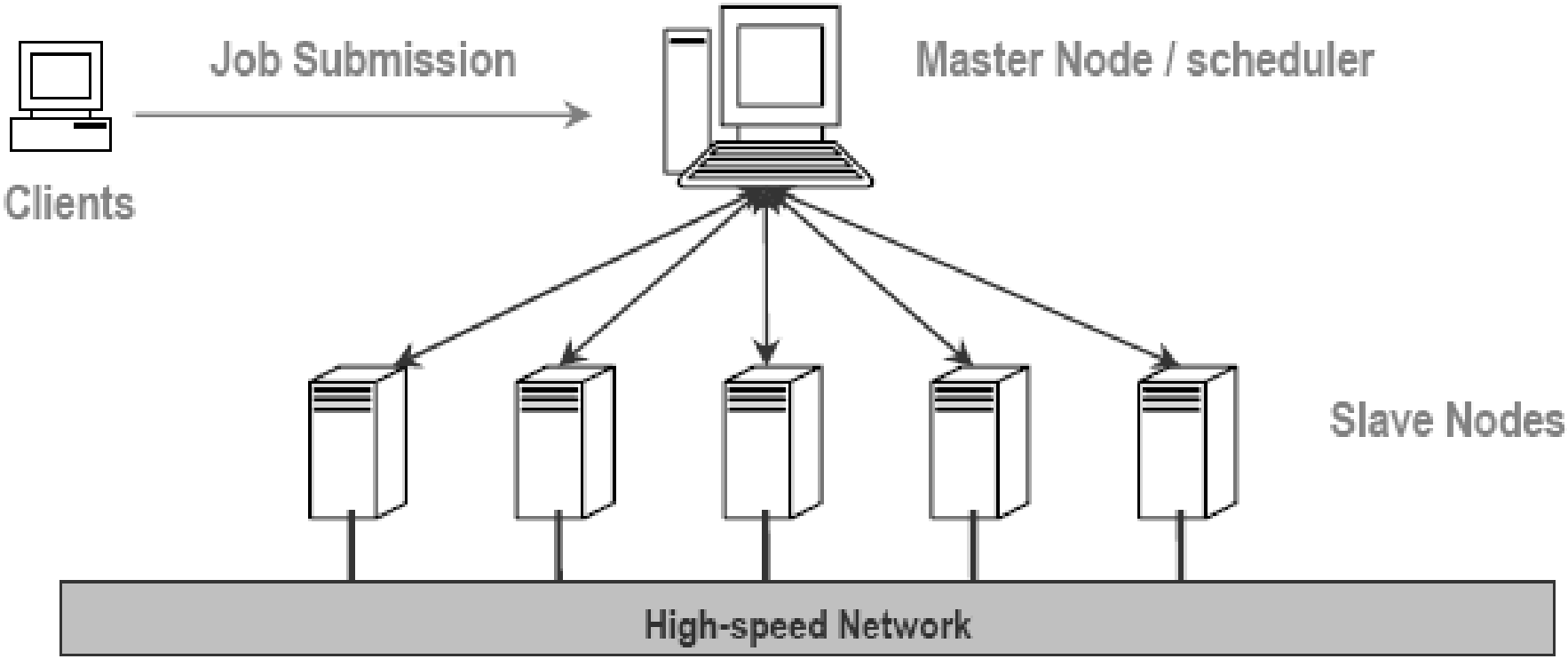
2009-2010

Curs 7

Planificare în Sisteme de Tip Cluster

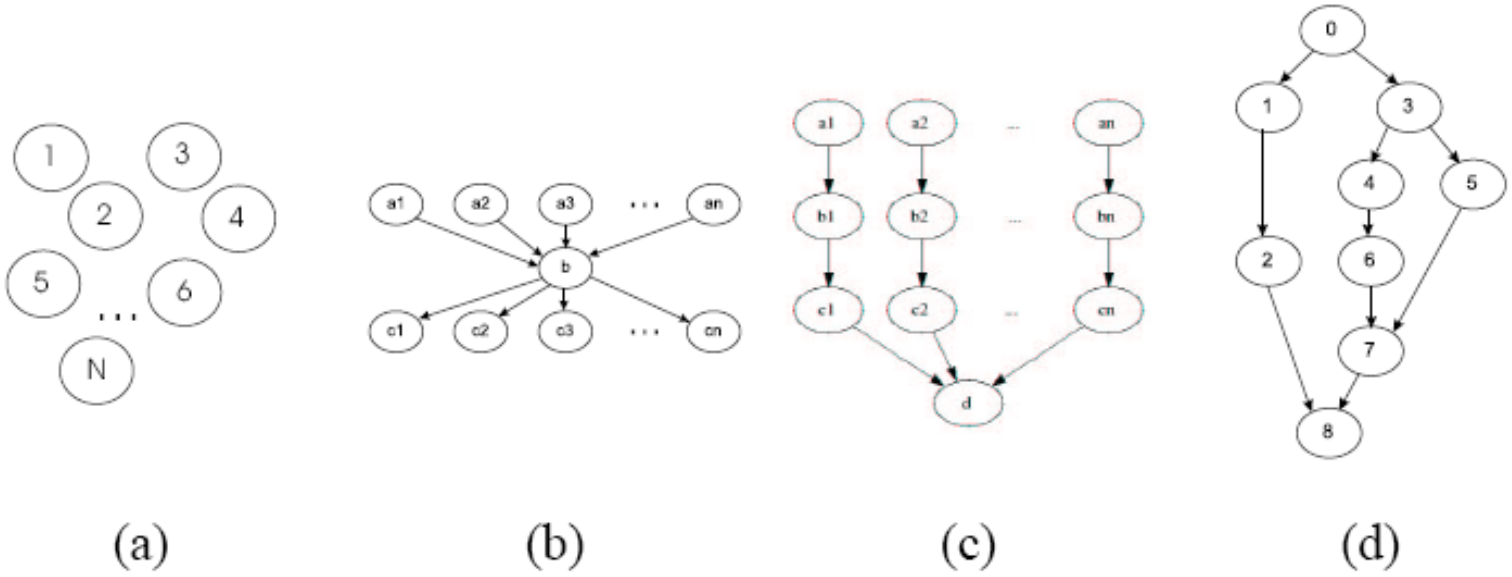
Cluster Scheduling

- **P,Q | pi; prmt; prec | Cmax**



Task Classification

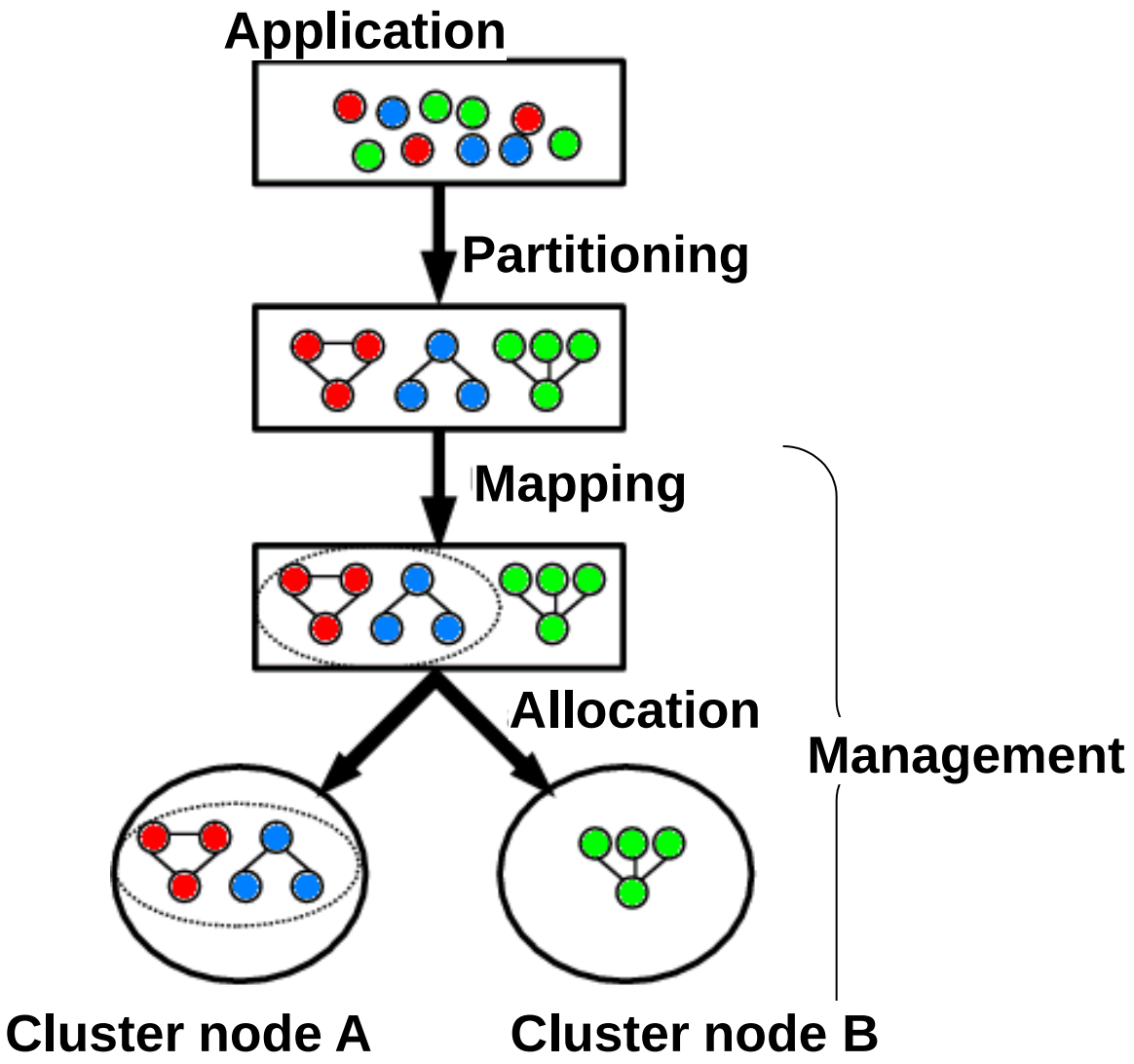
- Independent tasks (a)
- Loosely-coupled tasks (b,c)
- Tightly-coupled tasks (d)



(a) (b) (c) (d)

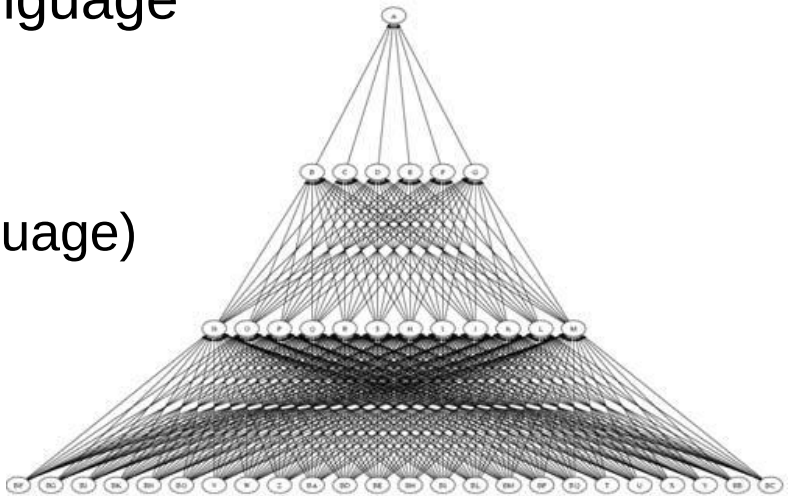
Application Management

- Description
- Partitioning
- Mapping
- Allocation



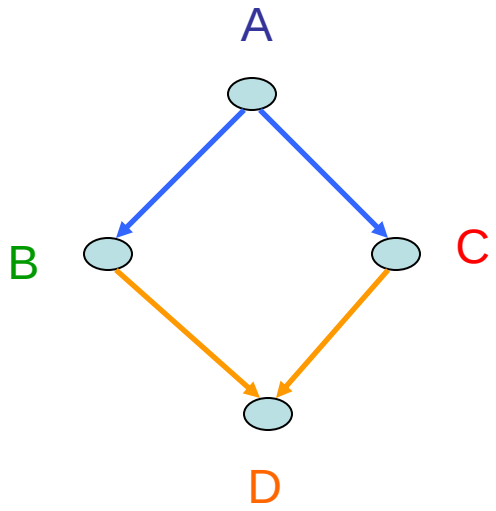
Task Description

- Use a grid application description language
  - VDL (Virtual Data Language)
  - Condor DAGMan
  - JSDL (Job standard description language)
  - GXML
  - AGWL
  - XPWSL
  - Grid-ADL and GEL
    - One can take advantage of loop construct to use compilation mechanisms for vectorization
  - JDL (Job description language)
  - …



Languages: Condor DAGMan

```
#
# first_example.dag
#
Job A A.condor
Job B B.condor
Job C C.condor
Job D D.condor
PARENT A CHILD B C
PARENT B C CHILD D
```



Languages: DAGMan

```
#
# task A
#
executable = A.exe
input = test.data
output = A.out
log = dag.log
Queue
```

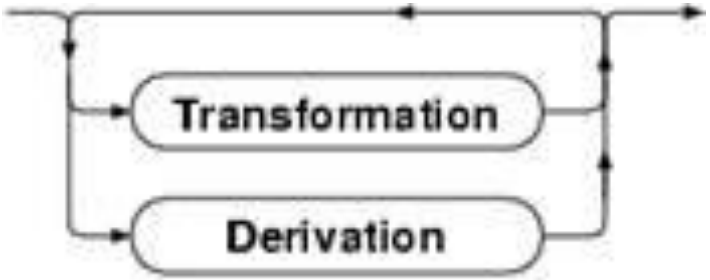
```
#
# task B
#
executable = B.exe
input = A.out
output = B.out
log = dag.log
Queue
```

```
#
# task C
#
executable = C.exe
input = A.out
output = C.out
log = dag.log
Queue
```

```
#
# task D
#
executable = D.exe
input = B.out C.out
output = final.out
log = dag.log
Queue
```

VDL (Virtual Data Language)

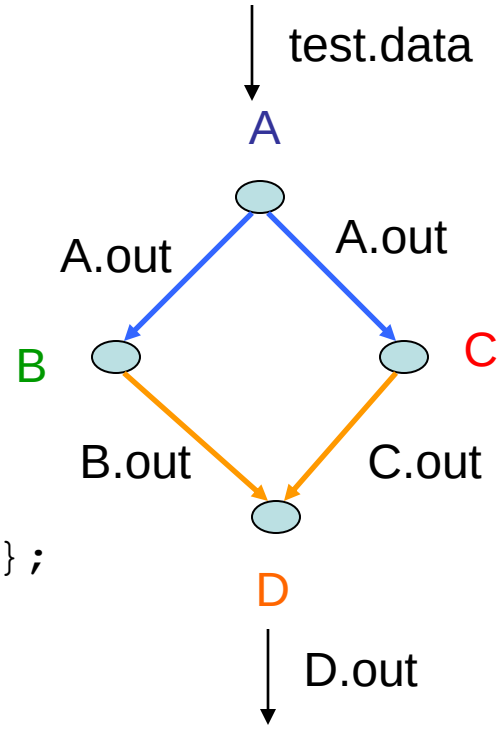
- Graph generated by Pegasus (Planning for Execution in Grids) through the analysis of the TRs and DVs
- Control is given to Condor DAGMan, once the graph is generated



http://vds.uchicago.edu/vds/doc/userguide/html/H_VDLReference.html

Languages: VDL

```
TR calculate{ output b, input a} {
        app vanilla = "calculate.exe";
        arg stdin = ${output:a};
        arg stdout = ${output:b};
}
TR analyze{ input a[], output c} {
        app vanilla = "analyze.exe";
        arg files = ${:a};
        arg stdout = ${output:c};
}
DV calculate { b=@{output:A.out},
                a=@{input:test.data} };
DV calculate { b=@{output:B.out},
                a=@{input:A.out} };
DV calculate { b=@{output:C.out},
                a=@{input:A.out} };
DV analyze{ a=[ @{input:B.out},
                   @{input:C.out} ],
             c=@{output:D.out} };
```



Languages: GXML

Part of the GANGA framework

(Grid Application iNformation Gathering and Accessing)

```
<flow:Workflow flow:start="A">
   <jsdl:Job jsdl:id="A">
   ...
   </jsdl:Job>
   <jsdl:Job jsdl:id="B">
   <flow:Depend flow:success="A"/>
   ...
   </jsdl:Job>
   <jsdl:Job jsdl:id="C">
   <flow:Depend flow:success="A"/>
   ...
   </jsdl:Job>
   <jsdl:Job jsdl:id="D">
   <flow:Depend flow:success="B & C"/>
   ...
   </jsdl:Job>
</flow:Workflow>
```

flow:LoopCount tag allows loops

Converted to a DAG and managed by Condor DAGMan

Languages: AGWL (Abstract Grid Workflow Language)

```
<agwl-workflow>
   <activity name="A" type="teste:A">
        <dataIn name="test.dat" > </dataIn>
        ...
        <dataOut name="A.out"> </dataOut>
   </activity>
      <subWorkflow name="tasksBandC">    // defining tasks B and C
   <body>
        <activity name="B" type="teste:B">
                <dataIn name="A.out" > </dataIn>
                ...
                <dataOut name="B.out"> </dataOut>
        </activity>
        <activity name="C" type="teste:C">
                <dataIn name="A.out" > </dataIn>
                ...
                <dataOut name="C.out"> </dataOut>
        </activity>
   </body>
     </subWorkflow>
   <activity name="D" type="teste:D">
        <dataIn name="B.out" > </dataIn>
        <dataIn name="C.out" > </dataIn>
        ...
        <dataOut name="D.out"> </dataOut>
   </activity>
</agwl-workflow>
```

Converted to CGWL
(Concrete Grid...)
and executed by ASKALON

Languages: XPWSL
XML-based Parallel Workflow Specification Language

```
<header>
   <name>DAG example</name>
   <description>same example
   previously used</description>
</header>

<assignment>
   <task id="T0" delay="delay_A">
        <path>/path</path>
        <code>A.exe</code>
   </task>
   <task id="T1" delay="delay_B">
        <path>/path</path>
        <code>B.exe</code>
   </task>
   <task id="T2" delay="delay_C">
        <path>/path</path>
        <code>C.exe</code>
   </task>
   <task id="T3" delay="delay_D">
        <path>/path</path>
        <code>D.exe</code>
   </task>
</assignment>
```

```
<datalink>
        <block type="sequential">
            <task_id>T0<task_id>
            <multi>1</multi>
            <input>test.dat</input>
        </block>
        <block type="sequential">
            <task_id>T1<task_id>
            <multi>1</multi>
            <input>A.out</input>
            <task_id>T2<task_id>
            <multi>1</multi>
            <input>A.out</input>
        </block>
        <block type="sequential">
            <task_id>T3<task_id>
            <multi>1</multi>
            <input>B.dat</input>
            <input>C.dat</input>
        </block>
</datalink>
```

Languages: GEL
Grid Execution Language

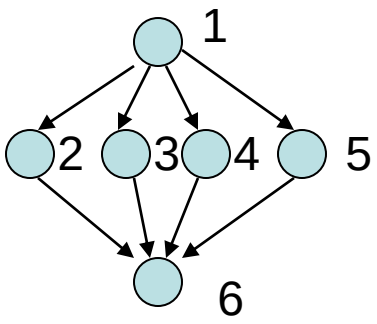
```
taskA = {exec="A.exe"; dir="/path";
   args="test.dat"}

taskB = {exec="B.exe"; dir="/path"; args="A.out"}

taskC = {exec="C.exe"; dir="/path"; args="A.out"}

taskD = {exec="D.exe"; dir="/path";
   args="B.out","C.out"}

taskA; taskB | taskC; taskD
```

Languages: GRID-ADL
Grid Application Description Language

```
graph loosely-coupled
task A -e A.sub -i data.in -o a.out
task B -e B.sub -i a.out -o b.out
task C -e C.sub -i a.out -o c.out
task D -e D.sub -i b.out c.out -o data.out
```

Languages: GRID-ADL
a more complex example

```
1  graph loosely-coupled
2  OUTPUT = "out1.png"
3  gnuplot= "/usr/local/bin/gnuplot"
4  task 1 -e ${gnuplot} -a "1.txt" -i in1.dat
5       -o out1.png
6  foreach TASK in 2..5 {
7     task ${TASK} -e ${gnuplot} -a ${TASK}".txt"
8          -i in${TASK}.dat -o out${TASK}.png
9     OUTPUT = ${OUTPUT} + ";out"+${TASK}+".png"
10 }
11 task 6 -e prepare_print -i ${OUTPUT} -o data.ps
12 transient ${OUTPUT}
```



Languages

Lang	Mw	RMS	Type	Wflw	DAG	DAGInf
DAGMan	DM	C	plain	no	yes	manual
VDL	DM	C	plain	yes	yes	auto
GXML	C	C	XML	yes	yes	manual
AGWL	AS	Glob	XML	yes	yes	manual
XPWSL	Join	Join	XML	yes	yes	manual
GEL	Gel	Vars	script	no	yes	auto
GRID-ADL	AM	AM/PBS	script	no	yes	auto

C: Cluster, DM: DAGMan, AS: Askalon, AM: AppMan, Vars: SMP, SGE, LSF, PBS, or Globus

Partitioning/Clustering

- Application represented as a graph
  - Nodes: job
  - Edges: precedence
- Graph partitioning techniques:
  - Minimize communication
  - Increase throughput or speedup
  - Need good heuristics
- Graph Partitioning
  - Optimally allocating the components of a distributed program over several machines
  - Communication between machines is assumed to be the major factor in application performance
  - NP-hard for case of 3 or more terminals
- Clustering

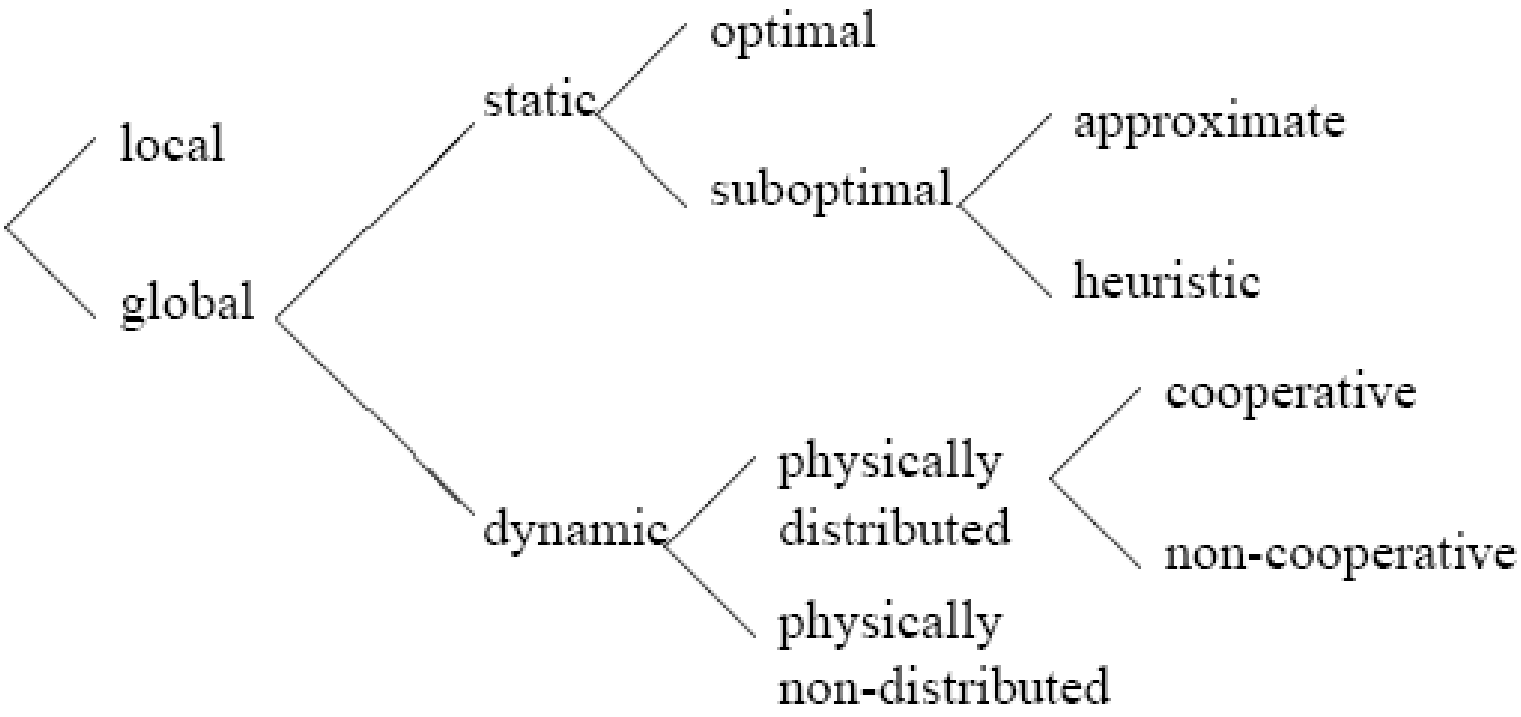
List Scheduling

- Make an ordered list of processes by assigning them some priorities
- Repeatedly execute the following two steps until a valid schedule is obtained:
  - Select from the list, the process with the highest priority for scheduling.
  - Select a resource to schedule this process.
- Priorities are determined statically before the scheduling process begins. The first step chooses the process with the highest priority, the second step selects the best possible resource.
- Some known list scheduling strategies:
  - **Highest Level First** algorithm (HLF)
  - **Longest Path** algorithm (LP)
  - **Longest Processing Time**
  - **Critical Path Method**
- List scheduling algorithms only produce good results for coarse-grained applications

Static scheduling task precedence graph DSC: Dominance Sequence Clustering

- **Yang and Gerasoulis, 1994**: two step method for scheduling with communication: (focus on the critical path)
  1) schedule an unbounded number of completely connected processors (cluster of tasks);
  2) if the number of clusters is larger than the number of available processors, then merge the clusters until it gets the number of real processors, considering the network topology (merging step).

Resource Management



Part of the Casavant and Kuhl's taxonomy

Resource Management

- The scheduling algorithm has four components:
  - **transfer policy**: *when* a node can take part of a task transfer;
  - **selection policy**: *which task* must be transferred;
  - **location policy**: *which node* to transfer to;
  - **information policy**: *when to collect* system state information.

- Location policy:
  - Sender-initiated
  - Receiver-initiated
  - Symetrically-initiated

Scheduling mechanisms for Grid

- Job scheduler
- Resource scheduler
- Application scheduler
- Meta-scheduler

Torque & Maui - USEFUL DEFINITIONS

- Batch systems provide a mechanism for *submitting*, *launching*, and *tracking* jobs on a shared resource.
- These services fulfil one of the major responsibilities of a batch system, providing *centralized access to distributed resources*.
- This allows users a ”single system image” in terms of the management of their jobs and the aggregate compute resources available.
- Furthermore, with a batch system, a scheduler is assigned the job of determining *when*, *where, and how* jobs are run so as to **maximize** the output of the cluster.

PBS & TORQUE

- PBS (Portable Batch System) - software system for managing system resources on workstations, SMP systems, MPPs and vector computers. It was based on Network Queuing System (NQS) 1986 NASA – first on Cray and then ported to other architectures

- TORQUE Resource Manager (Tera-scale Open-source Resource and QUEue manager) - an open source version of PBS, providing control over:
  - batch jobs
  - distributed compute nodes

TORQUE FEATURES

- TORQUE provides enhancements over standard OpenPBS in the following areas:
  - scalability
  - fault tolerance
  - scheduling interface
  - usability

TORQUE BENEFITS

- Initiate and manage serial and parallel batch jobs remotely (create, route, execute, modify and/or delete jobs)
- Define and implement resource policies that determine how much of each resource can be used by a job
- Apply jobs to resources across multiple servers to accelerate job completion time
- Collects information about the nodes within the cluster to determine which are in use and which are available.

PBS STRUCTURE

- General components (daemons)
  - A resource manager `pbs_server`
  - A scheduler `pbs_sched`
  - Many “executors”, moms (Machine Oriented Miniservers) `pbs_mom`

- PBS provide an API to communicate with the server and another one to interface the moms

PBS HANDLING A JOB

- User determines resource requirements(CPU time, memory, number of CPUs/node) for a job and writes a batch script.
- User submits job to PBS with the `qsub` command
- PBS places the job into a queue based on its resource requests and runs the job when those resources become available
- The job runs until it either completes or exceeds one of its resource request limits
- PBS copies the job's output into the directory from which the job was submitted and optionally notifies the user via email that the job has ended

PBS JOB SCRIPTS

- A PBS job script is just a regular shell script with some comments (the ones starting with #PBS) which are meaningful to PBS. These comments are used to specify properties of the job.

- Useful PBS options:

-l mem=N[KMG]	(request N [kilo|mega|giga] bytes of memory)
-l cput=hh:mm:ss	(max CPU time per job request)
-l walltime=hh:mm:ss	(max wall clock time per job request)
-l nodes=N:ppn=M	(request N nodes with M processors per node)
-S shell	(use *shell* instead of your login shell to interprete the batch script)
-j oe	(combine stdout and stderr together)

PBS JOB SCRIPTS(2)

- Here is a simple batch job:

```
#!/bin/bash
#PBS -l cput=00:00:40
#PBS -l mem=36MB
#PBS -l nodes=1:ppn=1
#PBS -N myjob
#PBS -j oe
#PBS -S /bin/ksh
cd $HOME/student/myjob/
/usr/bin/time ./mybin > myjob.out
```

- This job asks for one CPU on one node, 36MB of memory, and 40 seconds of CPU time. Its name is “myjob".

TIPICAL JOB SESSION

1) User submit job with qsub command
2) Server place job into exec queue and ask scheduler to examine job queues
3) Scheduler query moms for determining available resources (memory, cpu loadl, etc.)
4) Scheduler examine job queues, and eventually allocate resources for job, returning job id and resource list to server for execution
5) Server instruct mom to execute the commands section of the batch script
6) Mom execute batch commands, monitor resource usage of child processes and report back to server
7) Server email user notifying job end

USEFUL PBS COMANDS

- Starting the server, scheduler and mom deamons:

```
pbs_server
pbs_sched
pbs_mom
```

- Shuting down the server

```
qterm –t quick
```

- See all your available compute nodes:

```
pbsnodes -a
```

- Submitting a job:

```
qsub xyz.job
```

(Options can also be specified on the command line.)

- Monitoring a job

```
qstat -a
```

- Killing a job

```
qdel 3124
```

QMGR

- **Qmgr = manage policies and other batch configuration**
- **Qmgr comands:**

create => creates a new object, applies to queues and nodes.

delete =>is to destroy an existing object, applies to queues and nodes.

set =>is to define or alter attribute values of the object.

unset =>is to clear the value of attributes of the object. Note, this form does not accept an OP and value, only the attribute name.

list =>is to list the current attributes and associated values of the object.

print =>is to print all the queue and server attributes in a format that will be usable as input to the qmgr command.

PBS PROBLEMS

- The system sometimes hangs when some `pbs_mom` have problems
  Solution => Consider applying some patch

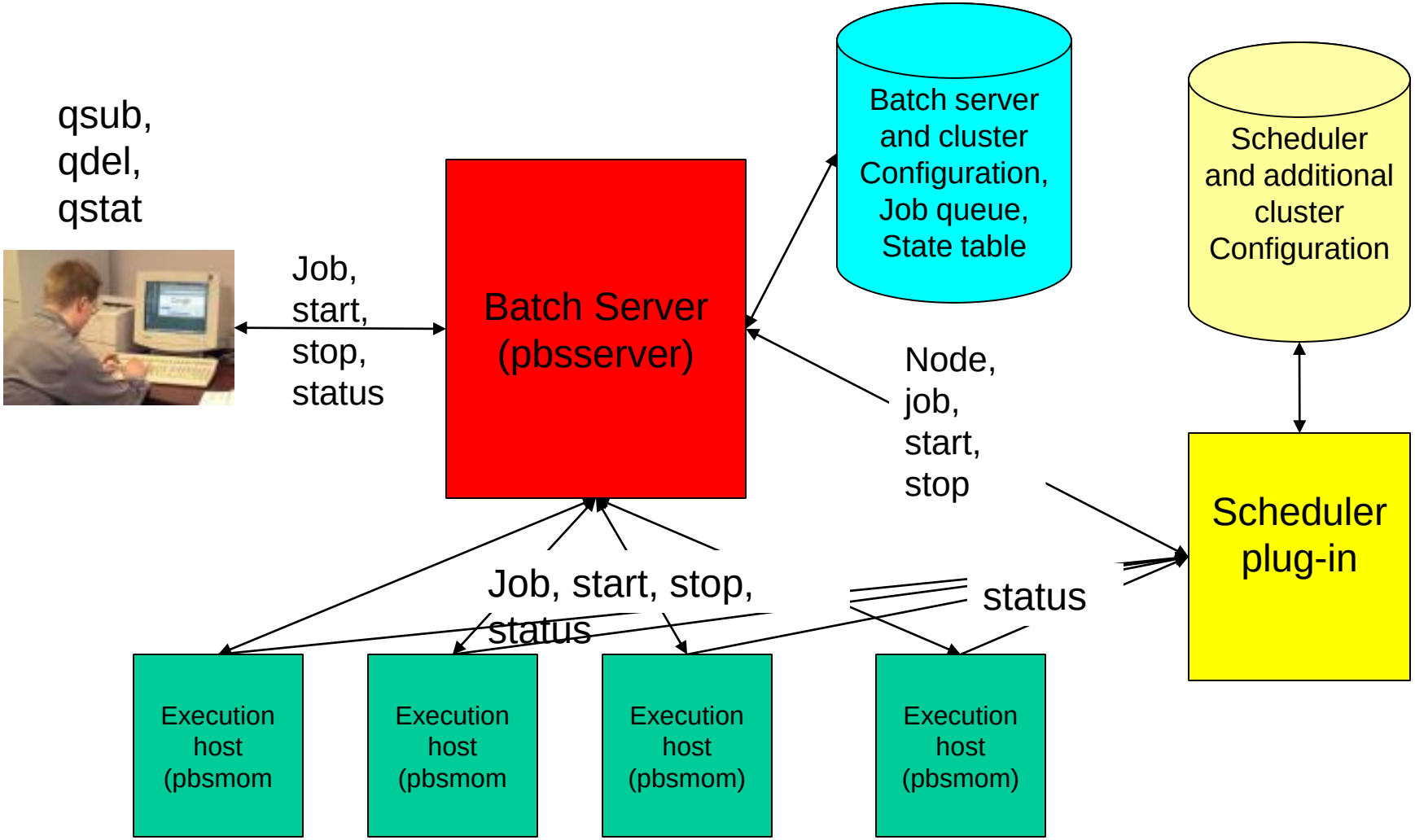
- Default PBS scheduler is an old-fashion scheduler, and lacks features like:
  - backfill scheduler
  - advanced reservation

- Solution => **MAUI** scheduler

Maui


Kauai
Lihue
Niihau
Kauai Channel
Oahu
HAWAII
Waipahu
Kaneohe
Honolulu
Kaiwi Chan.
Molokai
Kalaupapa
Lahaina
Kahului
Lanai
Maui
Kahoolawe
Alenuihaha Channel
PACIFIC
OCEAN
Waipio
Mauna Kea
Hilo
Kailua
Hawaii
Ka Lae
50 Miles
50 km

Batch System



MAUI

- **MAUI** is a scheduler (not a resource manager!) developed by the MAUI supercomputing center as an alternative to the default Loadleveler scheduler in their IBM SP environment

- Porting has been done to support also
  - OpenPBS
  - Wiki

- Porting has been done essentially using the appropriate API provided by the batch system.

SCHEDULING OBJECTS

- Maui functions by manipulating these five primary, elementary objects
  - Jobs
  - Nodes
  - Reservations
  - QOS(*Quality of service*) structures
  - Policies

SCHEDULING OBJECTS (2)

- A **job** consists of one or more requirements each of which requests a number of resources of a given type.
- A **node** is a collection of resources with a particular set of associated attributes
- **Policies** are generally specified via a config file and serve to control how and when jobs start.
- The Maui Scheduler allows administrators fine grain control over **QOS** levels on a per user, group and account basis.

MAUI FEATURES

- Backfill **scheduling algorithm**
- Priority scheduling algorithm
- Fair-share scheduling algorithm
- Reservations for high priority jobs
- More control parameters on users
- Commands for querying the scheduler

BACKFILL ALGORITHM

- Allows some jobs to be run 'out of order' so long as they do not delay the highest priority jobs in the queue
- Uses wallclock limit = an estimation of the wall time (or elapsed time) from job start to job finish to find “holes”(amounts of time dedicated to a job that has already finished or is waiting); it fills these “holes” with execution of other jobs in the queue
- The command **`showbf`** allows users to see exactly what resources are available for immediate use
- Better estimates of wallclock limit will increase the amount of improvement backfill scheduling can provide for your jobs

PRIORITY ALGORITHM

- Default is trivial FIFO but is weighted and combined based on service, requested resources, fair-share, direct priority, target, and bypass

```
Priority = serviceweight * servicefactor +
        resourceweight * resourcefactor +
        fairshareweight * fairsharefactor +
        directspecweight * directspecfactor +
        targetweight * targetfactor +
        bypassweight * bypassfactor
```

- Each *weight value is a configurable parameter and each *factor is calculated from subcomponents (I.e. user, group, priority, QoS etc.)

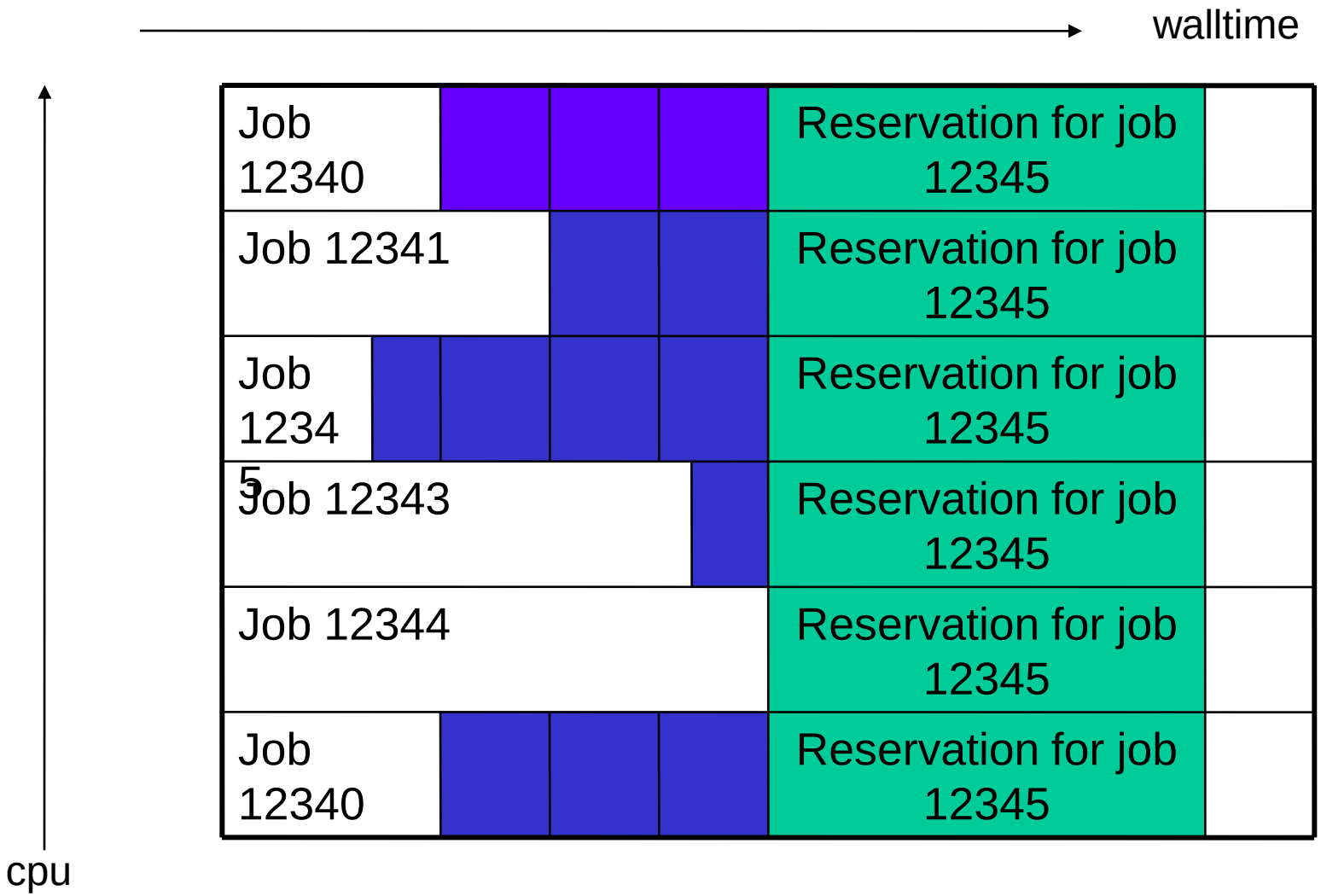
FAIR-SHARE ALGORITHM

- Composed of several parts which handle historical information, fairshare windows, usage, and impact
- All are site configurable parameters
- Purpose of a fairshare algorithm is to steer existing workload

ADVANCE RESERVATIONS

- Allow a site to set aside certain resources for specific uses over a given timeframe.
- Access control list (ACL): determines who or what can use the reserved resources.
- While reservation ACL's *allow* particular jobs to utilize reserved resources, they do not *force* the job to utilize these resources. Maui will attempt to locate the best possible combination of available resources whether these are reserved or unreserved.

Reservations



MAUI ADMIN COMANDS

- **`checknode` *`node_name`***
  Check node status

- **`diagnose, diagnose -p`**
  Show diagnostic info regarding various aspects, in particular jobs and job priorities

- **`showstats`**
  Show **MAUI** statistics

- **`showgrid stat_type`**
  Show a text-based table of **MAUI** statistics relative to *stat_type*

MAUI USER COMANDS

- **`showq`**
  Show job status and some job info

- **`showbf, showbf -v`**
  Check for immediately available CPUs and nodes

- **`checkjob` *`job_id`***
  Check job status

- **`canceljob` *`job_id`***
  Cancel a job, sending essentially a qdel to the pbs_server

- **`showstart` *`job_id`***
  Show when job is scheduled to start

Exam's quizzes

- **1.** Descrieţi ciclul de management al unei aplicaţii în procesul de execuţie pe un cluster.
- **2.** Enumeraţi cel puţin trei caracteristici pentru limbajele cu ajutorul cărora sunt descrise activităţile în procesul de planificare.
- **3.** Descrieţi pe scurs modelul de List Scheduling.
- **4.** Care sunt politicile de management a resurselor la nivel de cluster?
- **5.** Descrieți pe scurt principiul de funcționare PBS.
- **6.** Cum sunt definite politicile de planificare în MAUI?