Introduction

- Middleware systems are the cornerstone of any Grid or Cloud platform
- They operate at different levels
- From the lower levels close to the resources (processors, file systems, instruments) …
- … to the application themselves
- Several approaches and tools are now available, both in the public domain and sold by companies

- This presentation will not be exhaustive !!
Architecture of a Grid

Basic Grid Services
- Resource Discovery
- Resource Scheduling
- Uniform Computing Access
- Uniform Data Access
- Monitoring and Events
- Authentication
- Identity Credentials

Grid Services (transport services, security services)
- space-based networks
- optical networks
- Internet

Advanced Services
- Workflow management
- Fault management
- Grid MPI
- CORBA, DCOM, ...

Resource access and functionality
- Scientific instruments
- Clusters
- Condor pools of workstations
- National supercomputer facilities

Distributed Resources
- job initiation, event generators, GridFTP servers

The Grid Architecture Picture

User Portals
- Grid Access & Info
- Problem Solving Environments
- Application Science Portals

Service Layers
- Co-Scheduling
- Resource Discovery & Allocation
- Fault Tolerance
- Authentication
- Naming & Files
- Events
- Data Management

Resource Layer
- Computers
- Data bases
- Online instruments
- Software

Credit: Jack Dongarra
GridRPC Systems

http://graal.ens-lyon.fr/DIET/

RPC and Grid-Computing: GridRPC

- **One simple idea**
  - Implementing the RPC programming model over the grid
  - Using resources accessible through the network
  - Mixed parallelism model (data-parallel model at the server level and task parallelism between the servers)

- **Features needed**
  - Load-balancing (resource localization and performance evaluation, scheduling),
  - IDL,
  - Data and replica management,
  - Security,
  - Fault-tolerance,
  - Interoperability with other systems,
  - ...

- **Design of a standard interface**
  - within the OGF (GridRPC and SAGA WG)
  - Existing implementations: NetSolve, Ninf, DIET, OmniRPC
**DIET’s Goals**

- **Our goals**
  - To develop a toolbox for the deployment of environments using the Application Service Provider (ASP) paradigm with different applications
  - Use as much as possible public domain and standard software
  - To obtain a high performance and scalable environment
  - Implement and validate our more theoretical results
    - Scheduling for heterogeneous platforms, data (re)distribution and replication, performance evaluation, algorithmic for heterogeneous and distributed platforms, ...
  - Based on CORBA, NWS, LDAP, and our own software developments
    - FAST for performance evaluation,
    - LogService for monitoring,
    - VizDIET for the visualization,
    - GoDIET for the deployment

- **Based on** DIET Dashboard
  - Several applications in different fields (simulation, bioinformatics, ...)
- **Release 2.2 available on the web**
- **ACI Grid ASP, RNTL GASP, ANR LEGO, ANR Gwendia**
Some Research Topics

- **Scheduling**
  - Distributed scheduling
  - Software platform deployment with or without dynamic connections between components
  - Plug-in schedulers
- **Data-management**
  - Scheduling of computation requests and links with data-management
  - Replication, data prefetching
  - Workflow scheduling
- **Performance evaluation**
  - Application modelization
  - Dynamic information about the platform (network, clusters)
Data/replica management

• Two needs
  • Keep the data in place to reduce the overhead of communications between clients and servers
  • Replicate data whenever possible

• Three approaches for DIET
  • DTM (LIFC, Besançon)
    ▪ Hierarchy similar to the DIET’s one
    ▪ Distributed data manager
    ▪ Redistribution between servers
  • JuxMem (Paris, Rennes)
    ▪ P2P data cache
  • DAGDA (IN2P3, Clermont–Ferrand)
    ▪ Joining task scheduling and data management

• Work done within the GridRPC Working Group (OGF)
  • Relations with workflow management

DAGDA

Data Arrangement for Grid and Distributed Applications

• A new data manager for the DIET middleware providing
  • Explicit data replication: Using the API.
  • Implicit data replication: The data are replicated on the selected SeDs.
  • Direct data get/put through the API.
  • Automatic data management: Using a selected data replacement algorithm when necessary.
    ▪ LRU: The Least Recently Used data is deleted.
    ▪ LFU: The Least Frequently Used data is deleted.
    ▪ FIFO: The « oldest » data is deleted.
  • Transfer optimization by selecting the more convenient source.
    ▪ Using statistics on previous transfers.
  • Storage resources usage management.
    ▪ The space reserved for the data is configured by the user.
  • Data status backup/restoration.
    ▪ Allowing to stop and restart DIET, saving the data status on each node.
DAGDA

- **Transfer model**
  - Uses the pull model.
  - The data are sent independently of the service call.
  - The data can be sent in several parts.

1: The client sends a request for a service.
2: DIET selects some SeDs according to the chosen scheduler.
3: The client sends its request to the SeD.
4: The SeD downloads the data from the client and/or from other nodes of DIET.
5: The SeD performs the call.
6: The persistent data are updated.

FAST: Fast Agent’s System Timer

- Performance evaluation of platform enables to find an efficient server (redistribution and computation costs) without testing every configuration ➔ performance database for the scheduler
- Based on NWS (*Network Weather Service*)
DIET Plug-in Schedulers

• **SeD level**
  - Performance estimation function
  - Estimation Metric Vector (*estVector_t*) – dynamic collection of performance estimation values
    - Performance measures available through DIET
      - FAST-NWS performance metrics
      - Time elapsed since the last execution
      - CoRI (Collector of Resource Information)
    - Developer defined values
  - Standard estimation tags for accessing the fields of an *estVector_t*
    - EST_FREEMEM
    - EST_TCOMP
    - EST_TIMESSINCELASTSOLVE
    - EST_FREECPU

• **Aggregation Methods**
  - Defining mechanism how to sort SeD responses: associated with the service and defined at SeD level
  - Tunable comparison/aggregation routines for scheduling
  - Priority Scheduler
    - Performs pairwise server estimation comparisons returning a sorted list of server responses;
    - Can minimize or maximize based on SeD estimations and taking into consideration the order in which the request for those performance estimations was specified at SeD level.

DIET Scheduling

• **Collector of Resource Information (CoRI)**
  - Interface to gather performance information
  - Functional requirements
    - Set of basic metrics
    - One single access interface
  - Non-functional requirements
    - Extensibility
    - Accuracy and latency
    - Non-Intrusiveness

• **Currently 2 modules available**
  - CoRI Easy
  - Fast
  - Extension possibilities: Ganglia, Nagios
  - R-GMA, Hawkeye, INCA, MDS, …

---

**CoRI Manager**

- **CoRI-Easy Collector**
- **FAST Collector**
- **Other Collectors like Ganglia**
Workflow Management

- **Workflow representation**
  - Direct Acyclic Graph (DAG)
    - Each vertex is a task
    - Each directed edge represents communication between tasks

- **Goals**
  - Build and execute workflows
  - Use different heuristics to solve scheduling problems
  - Extensibility to address multi-workflows submission and large grid platform
  - Manage heterogeneity and variability of environment

**Architecture with MA DAG**

- Specific agent for workflow management (MA DAG)
- Two modes:
  - MA DAG defines a complete scheduling of the workflow (ordering and mapping)
  - MA DAG defines only an ordering for the workflow execution, the mapping is done in the next step by the client which pass by the Master Agent to find the server where execute the workflow services.
Platform Deployment

- **Problem**: mapping a middleware across many remote resources
- **Deployment phases**
  - Resource discovery, planning, resource selection, remote file installation, pre-configuration, launch, post-configuration
- And all this as automatically as possible

- Our objective: find an optimal deployment of agents and servers onto a set of (dedicated) resources
  - An **optimal** deployment is a deployment that provides the maximum throughput
  - $\rho$ is the throughput of the platform measured in number of requests completed per second

Deployment Management

**Diagram:**
- Distributed deployment of DIET
- XML:
  - Resources
  - Machines
  - Storage
  - DIET hierarchy
Optimal Deployment

Objective: find an optimal deployment of agents and servers for a set of resources $V$
- Maximum throughput $\rho$ of completed requests per second
  - maximizing the steady-state throughput (req/s)
  - Scheduling request throughput $\rho_{sched}$
  - Service throughput $\rho_{service}$

Lemma 1: The completed request throughput of a deployment is given by the minimum of the scheduling request throughput ($\rho_{sched}$) and the service request throughput ($\rho_{service}$)
$$\rho = \min(\rho_{service}, \rho_{sched})$$

Lemma 2: The scheduling throughput $\rho_{sched}$ is limited by the throughput of the agent with the highest degree

Lemma 3: The service request throughput $\rho_{service}$ increases as the number of servers included in a deployment increases


Complete Spanning D–ary Tree

- A **complete d-ary tree** is a tree in which every level, except possibly the deepest, is completely filled.
  
  All internal nodes except one have a degree, or number of children, equal to $d$; the remaining internal node is at depth $n-1$ and may have any degree from 1 to $d$.

- A **spanning tree** is a connected, acyclic subgraph containing all the vertices of a graph.

- A **complete spanning d-ary tree** (CSD tree) is a tree that is both a complete d-ary tree and a spanning tree.
**dMax Set**

- dMax set is the set of all trees for which maximum degree is equal to dMax
  - Examples: 3 trees from dMax set 4 and dMax set 6

**Optimal Deployment – Theorems**

- **Theorem:** the optimal throughput \( \rho \) of any deployment with maximum degree dMax is obtained with a CSD tree.
  - By Lemma 1 \( \rho = \min(\rho_{\text{service}}, \rho_{\text{sched}}) \)
  - By Lemma 2 \( \rho_{\text{sched}} \) is limited by agent with maximum degree
  - By Lemma 3 \( \rho_{\text{service}} \) increases with \( \text{card}(S) \)

- **Theorem:** The complete spanning d-ary tree with degree \( d \in [1, |V| - 1] \) that maximizes the minimum of the scheduling request and service request throughputs is an optimal deployment
  - Test all possible degrees \( d \in [1, |V| - 1] \)
  - Select MAX \( \min(\rho_{\text{service}}, \rho_{\text{sched}}) \)
Deployment construction for DIET

• $S_{\text{req}}$ is the size in MB of the message forwarded down the agent hierarchy for a scheduling request.
• $W_{\text{req}}$ is the amount of computation in MFlops needed by an agent to process one incoming request.
• $W_{\text{pre}}$ is the amount of computation in Mflops needed for a server to predict its own performance for a request.

Deployment construction for DIET

• $S_{\text{rep}}$ is the size in MB of the reply forwarded back up the agent hierarchy.
• $W_{\text{rep}}$ is the amount of computation in MFlops needed by an agent to merge the replies from its children.
• $W_{\text{app}}$ is the amount of computation in Mflops needed by a server to complete a service request for $\text{app}$ service.
Deployment constraints for an Agent

- Agent communication
  \[ \text{agent\_receive\_time} = \frac{S_{\text{req}} + dS_{\text{rep}}}{B} \]
  \[ \text{agent\_send\_time} = \frac{dS_{\text{req}} + S_{\text{rep}}}{B} \]

- Agent computation
  \[ \text{agent\_comp\_time} = \frac{W_{\text{req}} + W_{\text{rep}}(d)}{w} \]
  where \( W_{\text{rep}}(d) = W_{\text{fix}} + W_{\text{sel}} \cdot d \)

Deployment constraints for a Server

- Server communication
  \[ \text{server\_receive\_time} = \frac{S_{\text{req}}}{B} \]
  \[ \text{server\_send\_time} = \frac{S_{\text{rep}}}{B} \]

- Server computation
  \[ \text{all\_servers\_comp\_time} = \frac{W_{\text{pre}} \cdot |S| + W_{\text{app}}}{w} \]
  \[ \text{server\_comp\_time} = \frac{W_{\text{pre}} + W_{\text{app}}}{|S|} \]

\[
\begin{array}{ccc}
S1 & S2 & S3 \\
\vdots & \vdots & \vdots \\
W_{\text{app}1} & W_{\text{app}2} & W_{\text{app}3} \\
\end{array}
\]
Send or receive or compute, single port

• Steady state modeling

• Scheduling throughput

\[
\rho_{\text{sched}} = \min \left( \frac{1}{W_{\text{pre}} + \frac{S_{\text{req}}}{B} + \frac{S_{\text{rep}}}{B} + \frac{d_{\text{req}}}{B} + \frac{d_{\text{rep}}}{B} + \frac{W_{\text{req}} + W_{\text{rep}}[d]}{w}} \right)
\]

• Service throughput

\[
\rho_{\text{service}} = \frac{1}{S_{\text{req}} + S_{\text{rep}} + \frac{W_{\text{pre}} + W_{\text{app}}}{B}}
\]

\[\rho = \min(\rho_{\text{service}}, \rho_{\text{sched}})\]

• Greedy algorithms to construct the optimal tree for a given set of parameters

• The same model can be computed for (send || receive || compute)

Simulation and experimental design

• **Model parametrization**

<table>
<thead>
<tr>
<th>Components</th>
<th>(W_{\text{req}}) (Mflops)</th>
<th>(W_{\text{fix}}) (Mflops)</th>
<th>(W_{\text{sel}}) (Mflops)</th>
<th>(W_{\text{pre}}) (Mflops)</th>
<th>(S_{\text{rep}}) (MB)</th>
<th>(S_{\text{req}}) (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent</td>
<td>3.2x10^{-2}</td>
<td>1.0x10^{-3}</td>
<td>9.6x10^{-4}</td>
<td>–</td>
<td>6.4x10^{-3}</td>
<td>5.3x10^{-3}</td>
</tr>
<tr>
<td>SeD</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>6.4x10^{-3}</td>
<td>6.4x10^{-3}</td>
<td>5.3x10^{-3}</td>
</tr>
</tbody>
</table>

• **Experimentation setup**

  • **Software:** GoDIET is used for the deployment
  • **Job types:** DGEMM, a simple matrix multiplication (BLAS package)
  • **Workload:** steady-state load with 1 - 200 client scripts (each script launches requests serially)
  • **Resources:** dual AMD Opteron 246 processors @ 2GHz, each with cache size of 1024KB, 2GB of main memory and a 1Gb/s Ethernet
    - Lyon cluster - 55 nodes
    - Sophia cluster - 140 nodes
Throughput validation – Serial Model (DGEMM 10)

Throughput validation – DGEMM 1000, bandwidth 190 Mb/s
Summary

- Determine how many nodes should be used and design hierarchical organization
- Proved optimal deployment is a CSD tree
- Algorithm to construct optimal tree
- Deployment prediction is easy, fast and scalable
- Experiments validate model

<table>
<thead>
<tr>
<th>DGEMM size</th>
<th>Nodes</th>
<th>Best</th>
<th>Selected</th>
<th>Model</th>
<th>Star</th>
<th>Tri-ary</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>21</td>
<td>1</td>
<td>1</td>
<td>100%</td>
<td>22.4%</td>
<td>50.5%</td>
</tr>
<tr>
<td>100</td>
<td>25</td>
<td>2</td>
<td>2</td>
<td>100%</td>
<td>84.4%</td>
<td>84.6%</td>
</tr>
<tr>
<td>200</td>
<td>45</td>
<td>3</td>
<td>8</td>
<td>86.1%</td>
<td>40.0%</td>
<td>100%</td>
</tr>
<tr>
<td>310</td>
<td>45</td>
<td>15</td>
<td>22</td>
<td>98.5%</td>
<td>73.8%</td>
<td>74.0%</td>
</tr>
<tr>
<td>1000</td>
<td>21</td>
<td>20</td>
<td>20</td>
<td>100%</td>
<td>100%</td>
<td>65.3%</td>
</tr>
</tbody>
</table>
One Target Application

“"My team has created a very innovative solution, but we’re still looking for a problem to go with it.”"

DIET–BLAST

- **Basic Local Alignment Search Tool (BLAST)**
  - To find homologies between nucleotids or amino acids sequences.
- **Objectives**
  - To find clues about the function of a protein/gene comparing a newly discovered sequence to well-known ones.
- **Biological databases**
  - BLAST uses biological databases containing large sets of annotated sequences as entry parameter.
- **Usage**
  - Generally, biologists perform BLAST searches on large sets of sequences.

```
... A T C A A G T C ...
|     |     |     |     |     |
... A C C A - G T C ...
```
DIET-BLAST

• Each sequence of the entry set can be treated independently
  • A simple parallelization: To perform the searches for each sequence on a different node.
  • But efficient… A sequence weights at most some kilobytes and a search on it takes at most some minutes.

• Requirements to « gridify » BLAST
  • A way to submit and distribute the requests
    ⇒ A GridRPC middleware – DIET
  • To replicate the databases
    ⇒ DAGDA

⇒ A software to perform the sequences sets division and to merge the results: DIET-BLAST

DIET-BLAST

• DIET-BLAST execution
  • The user submit a large set of sequences and a database or a database identifier.
  • Entry set is divided.
  • The database is replicated on the platform using DAGDA.
  • DIET returns a set of computing resources.
  • Each sequence is treated on a different SeD.
  • Results merge into a single result file.
Parallelization and distribution of bioinformatics requests over the Grid

• **Context**
  - Large sets of requests are submitted by many users of the grid.
  - Large databases are used as parameter of the requests.
  - The computing nodes of the grid have different computing and storage capacities.

• The jobs have to be scheduled « on-line » (the scheduling is made job by job)
  - We want to optimize the resources usage:
    - No useless replication.
    - Platform throughput optimization.

Where to replicate the databases ?
How to distribute the requests ?

Parallelization and distribution of bioinformatics requests on the Grid

• Moreover, to obtain dynamic information about the nodes status is a difficult task involving complex grid services.

• The obtained information can be several minutes old
  - For long execution time jobs: Not really a problem.
  - For jobs that take some minutes: The information are outdated.

• In this context without more information, there are few things to do... Using the classical Minimum Completion Time scheduling strategy is a good way to schedule the jobs but...

• With few more information, we can do better
  - The analysis of the execution traces of bioinformatics clusters showed that the way the biologists are submitting jobs is homogeneous if the observed time interval is long enough. We will use this information to optimize the Grid resources usage.
Parallelization and distribution of bioinformatics requests on the Grid

- Problem modeling
  We will denote:
  - \( m_i \) is the server \( i \) storage capacity.
  - \( w_i \) is the server \( i \) computing capacity.
  - \( \delta^j_i = 1 \) if the data \( j \) is on the server \( i \)
    \( \delta^j_i = 0 \) otherwise.
  - \( n_i(k,j) \) is the number of requests of type \( k \) on the database \( j \) performed on the server \( i \).
  - \( \alpha_k \) and \( c_k \) are two constants characterizing the algorithm complexity.
  - \( f_{k,j} \) is the frequency of the request of type \( k \) on the database \( j \).

- Scheduling and Replication Algorithm (SRA)
  - Relies on the integer approximation of a linear program
  - Gives the data distribution and the jobs scheduling
  - Takes into account the specificities of the biologists submissions

Simultaneous Scheduling of Replication and Computation for Data-Intensive Applications on the Grid.

Scheduling and Replication Algorithm

SRA uses the following linear program to determine the data distribution \( \delta^j_i \) and how many requests of each type to execute on each computing node \( n_i(k,j) \). The value to optimize is the throughput \( TP \).

\[
\begin{align*}
\forall j \sum_{i=1}^{m} \delta^j_i & \geq 1 & \text{Data distribution} \\
\forall i \sum_{j=1}^{n} \delta^j_i \cdot \text{size}_j & \leq m_i & \text{Storage capacities} \\
\forall i \sum_{j=1}^{n} \sum_{k=1}^{p} n_i(k,j) (\alpha_k \cdot \text{size}_j + c_k) & \leq w_i & \text{Computing capacities} \\
\forall i\forall j\forall k \; n_i(k,j) & \leq \frac{\delta^j_i \cdot \frac{w_i}{\alpha_k \cdot \text{size}_j + c_k}} & \text{Number of jobs to execute} \\
\forall j \forall k \sum_{i=1}^{m} n_i(k,j) & = f_{k,j} \cdot TP & \text{Throughput}
\end{align*}
\]

- SRA gives good results when the requests frequencies are constant
  - The data distribution and the scheduling are efficient
  - It does not need dynamic information about the Grid
Scheduling and Replication Algorithm

- Simulation experiments
  - Grid’5000 topology
  - 2540 heterogeneous CPUs on 9 sites
  - 5 databases of different size
  - 5 algorithms of different complexity
  - 1 request per 0.3 s.
  - The data are randomly distributed before the start
- For small sets of requests
  - SRA does not have enough time to replicate the data
- For large sets of requests
  - The replications and jobs scheduling of SRA avoid the computing nodes to saturate

Using MCT

Using SRA

SRA with frequency variation detection

- On the grid, the number and variety of users can make the time needed to have constant frequencies very long. Some « events » can temporary modify the frequencies
  - Data challenges
  - Important conference submission deadline
  - Holidays and week-ends
  - …

By detecting such a frequency variation, we can correct the data distribution and jobs scheduling

Parallelization and Distribution Strategies of Large Bioinformatics Requests over the Grid.
• Algorithm principle

- SRA gives an initial data distribution
- The Resource Broker records the submissions and update the frequencies
- If a frequency varies beyond a \( \varepsilon \) threshold
  - New data distribution computation using SRA
  - Start of the transfers asynchronously if possible. Otherwise, the job scheduling will cause the data transfers.
- Task scheduling using the job distribution given by SRA.

\[
\begin{align*}
\text{init}(f_{k,i}) \\
\text{nb} & \leftarrow 0 \\
\delta^j_i, n_i(k,j) & \leftarrow \text{SRA}(f_{k,i}) \\
\text{While} (\text{The scheduler can receive new tasks } T_{k,j}) \ & \\
\text{Record } T_{k,j} \text{ in } f'_{k,i} \\
\text{If} (\text{nb} \geq N \text{ AND } \exists k, j \text{ such that } |f_{k,i} - f'_{k,i}| \geq \varepsilon) \\
\delta^j_i, n_i(k,j) & \leftarrow \text{SRA}(f_{k,i}) \\
\text{Redistribution}(\delta^j_i) \ (\text{if asynchronous transfers are possible}) \\
\text{nb} & \leftarrow 0 \\
\text{Re-init}(f'_{k,i}) \\
\text{EndIf} \\
\text{Task scheduling using } n_i(k,j) \\
\text{nb} & \leftarrow \text{nb} + 1 \\
\text{EndWhile}
\end{align*}
\]

SRA with frequency variation detection

- The frequencies vary – We do not proceed to data redistribution
  - The scheduling is no more efficient for large sets of jobs.
  - The average execution time of the jobs increases with the number of jobs submitted.

- The frequencies vary – We detect the variations and proceed to the data redistribution
  - The scheduling efficiency is saved.
  - The average execution time of the jobs does not increase too much.
DIET implementation

• The 2 SRA-based algorithms need a centralized scheduling
  ▷ DIET proposed only distributed scheduling

• They also need an efficient data management allowing
  asynchronous data transfers and replication
  ▷ In DIET, the only way to move a data was to call a service with
    this data as parameter
  ▷ DTM does not manage data replications

• We introduced:
  ▷ A new plugin scheduler system, dynamically loading a scheduling
    algorithm at the agents levels
  ▷ DAGDA as a new data manager for the DIET middleware

DIET implementation

• Agents level plugin scheduler
  ▷ The scheduler classes in DIET agents are organized as follows

• We introduced a dynamic scheduling class loading system
  ▷ Allowing to centralize the scheduling on the Master Agent
  ▷ Making SRA possible to be implemented in DIET
DIET implementation

• SRA implementation:
  - We pass the platform description to the scheduler module
  - The scheduler keeps a track of the submitted jobs
  - The SeD is selected using the SRA job distribution.

• Dynamic SRA implementation:
  - Like for the SRA scheduling but:
    ▪ The scheduler detects the frequencies variations
    ▪ It calls DAGDA to perform asynchronous replications of the data

The Dynamic version of SRA does not need to know the initial jobs frequencies. It will estimate them after the first submissions.

Job scheduling & data replication

• In « Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications », Foster et al. analysed the performances of various combinations of job scheduling and data replication algorithms.

• They evaluated four job scheduling strategies:
  ▪ JobRandom: The job is sent to a random site.
  ▪ JobLeastLoaded: The job is sent to the node with the least number of waiting jobs.
  ▪ JobDataPresent: The job is sent on the least loaded site on which the data is present.
  ▪ JobLocal: The job is run locally.

• And three replication strategies:
  ▪ DataDoNothing: The data are not replicated. Data may be downloaded from a remote site in a cache using LRU replacement algorithm.
  ▪ DataRandom: The most « popular » data are replicated on a random site.
  ▪ DataLeastLoaded: The most « popular » data are replicated on the least loaded site in the neighborhood of the node.
Job scheduling & data replication

• For the different combinations they measured:
  • Best average response time when:
    ▪ The job is scheduled where the data is
    ▪ With data replication (even with random replication)

  • Best average data transfered when:
    ▪ The job is scheduled where the data is
    ▪ With data replication

  • Best average idle time:
    ▪ The job is scheduled where the data is
    ▪ With data replication

⇒ Scheduling should take into account the data distribution.
⇒ It is not always necessary to couple data movement & computation.

Integration of Scheduling & Replication

• In « Integration of Scheduling and Replication in Data Grids », Chakrabati et al. present the Integrated Replication and Scheduling Strategy (IRS) which couple scheduling and replication strategies.
  • Improve the performances by working alternatively on the data mapping and the task mapping.

• Algorithm principle:
  • The job scheduling can use two approaches:
    ▪ One based on the data availability: The job is sent on the node which have the most of the needed data.
    ▪ One based on scheduling cost (time to transfer the data, job waiting time and job execution time): The job is sent on the node which have the smallest cost.
  • After each job submission, a data usage matrix is updated.
  • The replication algorithm have two phases:
    ▪ Estimation of the maximum number of replications for a data.
    ▪ The data replication itself using the data usage matrix
Conclusions and future work

- **GridRPC**
  - Interesting approach for several applications
  - Simple, flexible, and efficient
  - Many interesting research issues (scheduling, data management, resource discovery and reservation, deployment, fault–tolerance, ...)

- **DIET**
  - Scalable, open–source, and multi–application platform
  - Concentration on several issues like resource discovery, scheduling (distributed scheduling and plugin schedulers), deployment (GoDIET and GRUDU), performance evaluation (CoRI), monitoring (LogService and VizDIET), data management and replication (DTM, JuxMem and DAGDA)
  - Large scale validation on the Grid5000 platform
  - A middleware designed and tunable for an application given

- **And now …**
  - Validation using SeD_batch (Loadleveler version)
  - Data management optimization
  - Scheduling optimization
  - More information and statistics for the users
  - Fault tolerance mechanisms
  - DIET over Cloud platforms (Eucalyptus)

**Grid’5000**

http://www.grid5000.org/

Batch Systems
OAR: A polyvalent resource scheduler

- A polyvalent and tunable task and resource manager
  - Following the evolution of technologies
  - Tuned for different platforms (cluster-on-demand, virtual clusters, Grid'5000 like platforms, large clusters, …)

- 2 levels platform

General architecture

- Database with a central role
  - Internal state that can be easily accessed
  - Several little perl modules
  - Each module can be easily replaced
**OAR: Resource hierarchy**

- Adapted for new architectures (multi-clusters, NUMA architectures, multicore processors, ...)

![Resource property hierarchy diagram]

**Some features**

- Hierarchical resource requests (handle heterogeneous clusters)
- Gantt scheduling (so you can visualize the internal scheduler decisions)
- Full or partial time-sharing
- Checkpoint/resubmit
- Licences servers management support
- Best effort jobs: if another job wants the same resources then it is deleted automatically
- Environment deployment support (Kadeploy)
  
  [Link to Kadeploy website](http://kadeploy.imag.fr/)
Some features, cont.

- Batch and Interactive jobs
- Admission rules
- Walltime
- Multi-schedulers support
- Multi-queues with priority
- Backfilling
- First-Fit Scheduler
- Reservation
- Support of moldable tasks
- Check compute nodes
- Epilogue/Prologue scripts
- Support of dynamic nodes
- Logging/Accounting
- Suspend/resume jobs
- ...

Submission examples

- Interactive task submission
  - `oarsub -l nodes=4 -i`

- Batch submission (with walltime and queue choice)
  - `oarsub -q default -l walltime=2:00,nodes=10 /home/toto/script`

- Reservation submission
  - `oarsub -r "2008-04-27 11:00" -l nodes=12`

- Connection to a reservation (uses the task number)
  - `oarsub -C 154`
More complex submission examples

- `oarsub --checkpoint 600 --signal 2 -t idempotent ./prog.sh`

- `oarsub -t besteffort -t idempotent -l cpu=1 ./prog.sh`

- `oarsub -l nodes=20/core=1 'taktuk -c "oarsh" \ -f $OAR_FILE_NODES broadcast exec [ date ]'`

- `oarsub -l cpu=2,walltime=16 -l cpu=4,walltime=8 \ -l cpu=8,walltime=4 ./moldable_prog.sh`

- `oarsub -l "{mem = '32G' or mem = '64G'}/cpu=1+/cpu=32"`

- `oarsub -l \ -l "switch=2/nodes=10+{type = 'matlab'}/licence=20"

Scheduler organization

Using queues
- Each queue has a priority
- Each queue has its own scheduling policy
First-Fit policy (backfilling)

Filling holes if the tasks’ order is not modified

Fair Sharing

- The order is computed based on what has been consumed (low demand is encouraged). Definition of a window and weight parameters
Advance reservation

- Useful in some cases (demo, planification, grid tasks, ...)
- But
  - It raises scheduling problems
  - Waist of resources

Workflow Systems
Scientific workflows

- Capture individual data transformation and analysis steps
- Large monolithic applications broken down to smaller jobs
  - Smaller jobs can be independent or connected by some control flow/data flow dependencies
  - Usually expressed as a Directed Acyclic Graph of tasks (DAG)
- Allows the scientists to modularize their application
- Scaled up execution over several computational resources

Credits: Pegasus

Workflow lifecycle

Credits: Pegasus
Abstract Workflow

A reliable, scalable workflow management system that an application or workflow composition service can depend on to get the job done

Cyberinfrastructure: Local machine, cluster, Condor pool, OSG, TeraGrid

Credits: Pegasus

Comparison of abstract and executable workflows

<table>
<thead>
<tr>
<th>Abstract Workflow</th>
<th>Executable Workflow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Describes your workflow at a logical level</td>
<td>Describes your workflow in terms of physical files and paths</td>
</tr>
<tr>
<td>Site Independent</td>
<td>Site Specific</td>
</tr>
<tr>
<td>Captures just the computation that the user (you) want to do</td>
<td>Has additional jobs for data movement etc.</td>
</tr>
</tbody>
</table>

Credits: Pegasus
Pegasus WMS

Workflow Description in XML

Properties

Pegasus Workflow Mapper

Replica Catalog

Site Catalog

Transformation Catalog

Condor DAGMan

Condor Schedd

Submit Host

TeraGrid
Open Science Grid
Campus resources
Local machine

Pegasus WMS restructures and optimizes the workflow, provides reliability

Credits: Pegasus

Discovery

• Data
  • Where do the input datasets reside?
  • Data is replicated for better performance (use of replica catalogs)

• Executables
  • Where are the executables installed?
  • Do binaries exist somewhere that can be staged to remote grid sites?

• Site Layout
  • What does a grid site look like?
Basic Workflow Mapping

- Select where to run the computations
  - Change task nodes into nodes with executable descriptions
    - Execution location
    - Environment variables initializes
    - Appropriate command-line parameters set

- Select which data to access
  - Add stage-in nodes to move data to computations
  - Add stage-out nodes to transfer data out of remote sites to storage
  - Add data transfer nodes between computation nodes that execute on different resources

Pegasus Workflow Mapping

Original workflow: 15 compute nodes devoid of resource assignment

Resulting workflow mapped onto 3 Grid sites:

<table>
<thead>
<tr>
<th>Number of Nodes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Compute nodes (4 reduced based on available intermediate data)</td>
</tr>
<tr>
<td>13</td>
<td>Data stage-in nodes</td>
</tr>
<tr>
<td>8</td>
<td>Inter-site data transfers</td>
</tr>
<tr>
<td>14</td>
<td>Data stage-out nodes to long-term storage</td>
</tr>
<tr>
<td>14</td>
<td>Data registration nodes (data cataloging)</td>
</tr>
</tbody>
</table>

Credits: Pegasus
Workflow Restructuring to improve Application Performance

- Cluster small running jobs together to achieve better performance
- Why?
  - Each job has scheduling overhead
  - Need to make this overhead worthwhile
  - Ideally users should run a job on the grid that takes at least 10 minutes to execute

Credits: Pegasus
Current and Future Research

- Resource selection
- Resource provisioning
- Workflow restructuring
- Adaptive computing
  - Workflow refinement adapts to changing execution environment
- Workflow provenance (including provenance of the mapping process)
- Management and optimization across multiple workflows
- Workflow debugging
- Streaming data workflows
- Automated guidance for workflow restructuring
- Support for long-lived and recurrent workflows

Credits: Pegasus

Conclusion
Conclusion and Research Issues

• Many middleware systems available in open-source for clusters, Grids and now Clouds
• Several success stories connecting middleware systems together (Pegasus, DIET, Globus, Ninf, …)
• “Some” research issues still need to be fixed and some nice research problems to solve theoretically and validated on actual platforms
  - Models (from theoretical models to actual platforms)
  - Application level scheduling
  - Scalability
  - Transparent data management
  - Taking virtualization into account
  - Fault tolerance
  - Security
  - Energy consumption
  - …
• Links with Petascale and Exascale systems
  - Fault tolerance, energy consumption, resource management, …

Questions?