Semi-Static Data Replication and Request Scheduling

From algorithms to middleware and applications

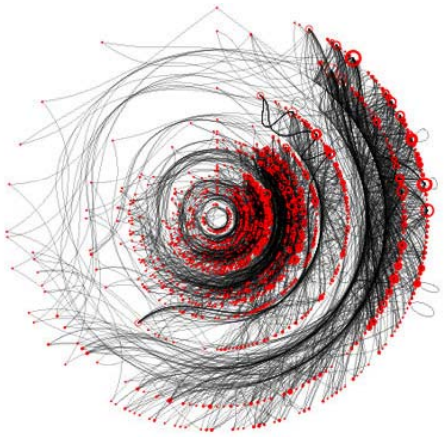
Frédéric Desprez

LIP ENS Lyon / INRIA

GRAAL Research Team

Join work with

E. Caron, G. Le Mahec and A. Vernois



CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

INRIA

UNIVERSITÉ DE LYON

Agenda

- Introduction
- One target application: BLAST using large databases over the grid
- One problem: join scheduling and replication
- One middleware : DIET/DAGDA
- Conclusion and future work

Introduction

- Several applications ready (and not only number crunching ones !)
- Huge data sets start to be available around the world
- Data management is one of the most important issue of today's application
- Replication has to be used to improve platform throughput

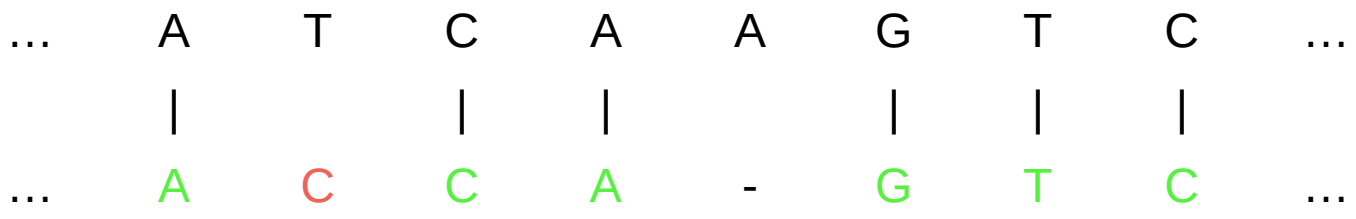
- Services for resource management and data management/replication available in most grid middleware …
- … but usually they work separately

- Our approach
  - Put the data management into the resource management sphere
  - Perform the request scheduling with the data replication based on the information provided by the application

- Application to a large scale bioinformatics application

One target application: BLAST over the grid

- 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.



One target application: BLAST over the grid

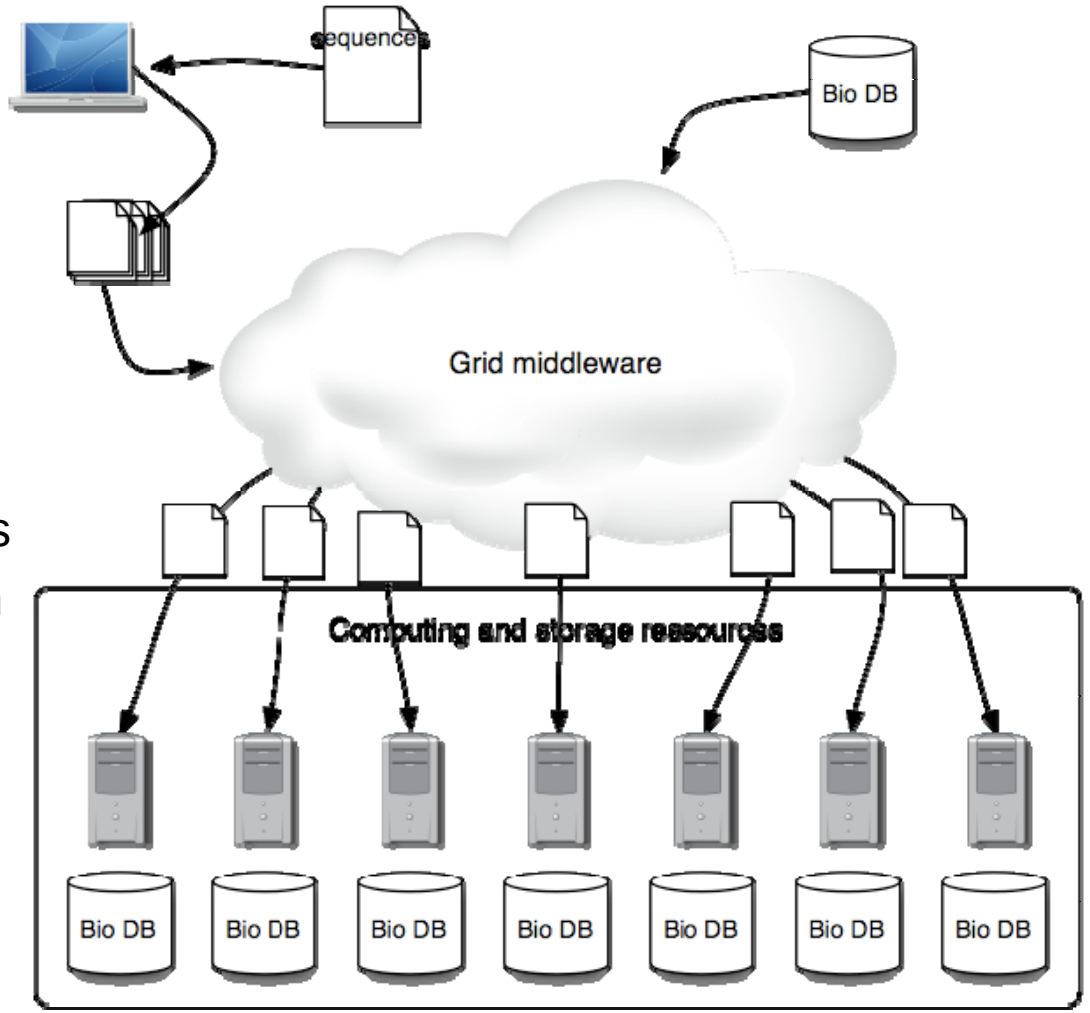
- Each sequence of the entry set can be treated independently
  - A simple parallelization: to perform the searches for each sequence on different nodes
  - But efficient… A sequence weights at most some kilobytes and a search on it takes at most some minutes

- Requirements to “gridify” BLAST applications
  - A way to submit and distribute the requests
  - A way to replicate databases

⇨ Then a software to perform
  ⇨ the sequences sets division before the computation
  ⇨ the results merge on exit

One target application: BLAST over the grid

- Grid-BLAST execution
  - The user submits a large set of sequences and a database or a database identifier
  - Entry set is divided.
  - The database is replicated on the platform using a data manager
  - The resource broker returns a set of computing resources
  - Each sequence is treated on a different server
  - Results merged 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 parameters of the requests
  - The computing nodes of the grid have different computing and storage capacities
  - Several different BLAST applications depending upon users' needs

- 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 ?

- Nice "little scheduling problem" (Yves Robert approved!)

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 information obtained 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 (MCT) 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.
- $\delta_i^j = 1$ if the data j is on the server i
  $\delta_i^j = 0$ otherwise.
- $n_i(k,j)$ is the number of requests of type k on the database j performed on the server i.
- α_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

Scheduling and Replication Algorithm

SRA uses the following linear program to determine the data distribution δ_i^j 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.

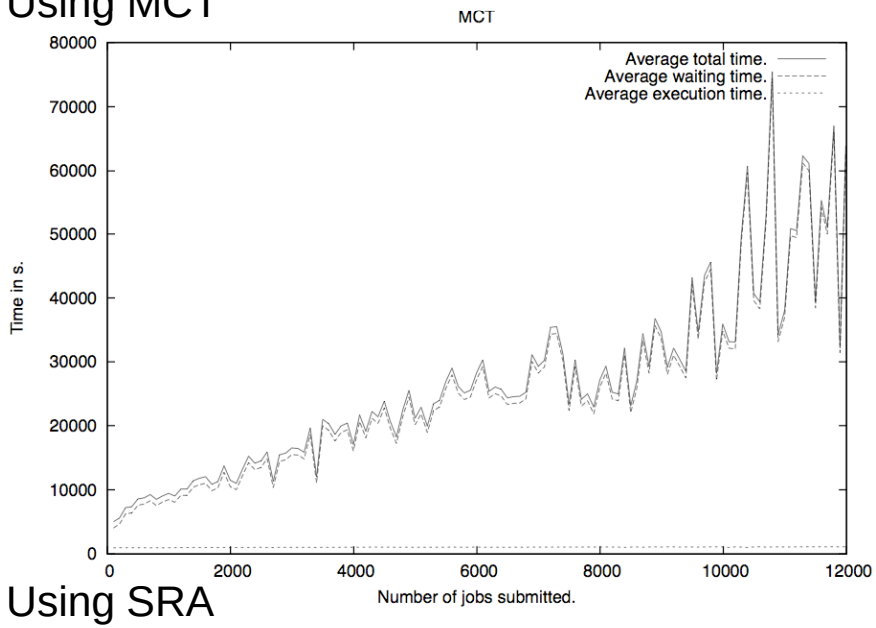
$$
\begin{cases}
\forall j \sum_{i=1}^{m} \delta_i^j \geq 1 & \text{Data distribution} \\
\forall i \sum_{j=1}^{n} \delta_i^j . size_j \leq m_i & \text{Storage capacities} \\
\forall i \sum_{k=1}^{p} \sum_{j=1}^{n} n_i(k,j)(\alpha_k . size_j + c_k) \leq w_i & \text{Computing capacities} \\
\forall i \forall j \forall k \; n_i(k,j) \leq \delta_i^j \frac{w_i}{\alpha_k . size_j + c_k} & \text{Number of jobs to execute} \\
\forall j \forall k \sum_{i=1}^{m} n_i(k,j) = f_{k,j} . TP & \text{Throughput}
\end{cases}
$$

- 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

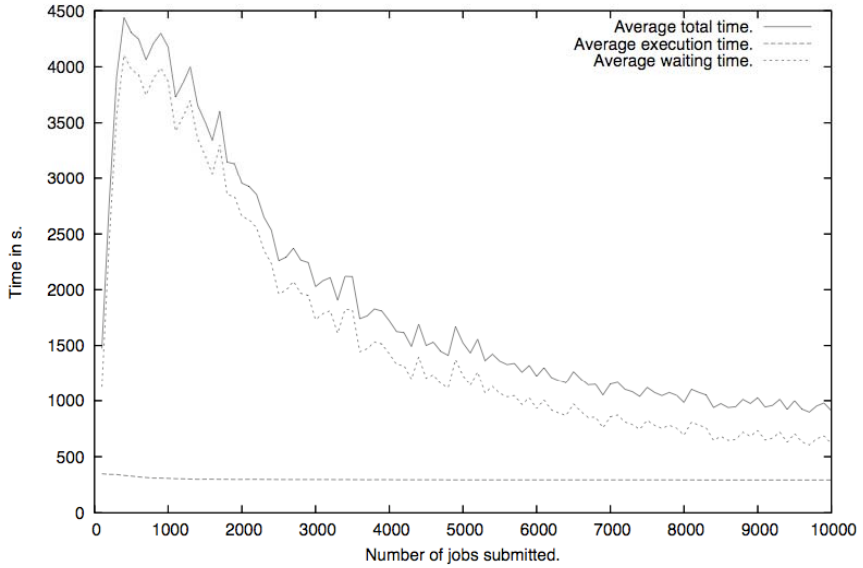
Scheduling and Replication Algorithm

- Simulation experiments
  - Grid'5000 platform
  - 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

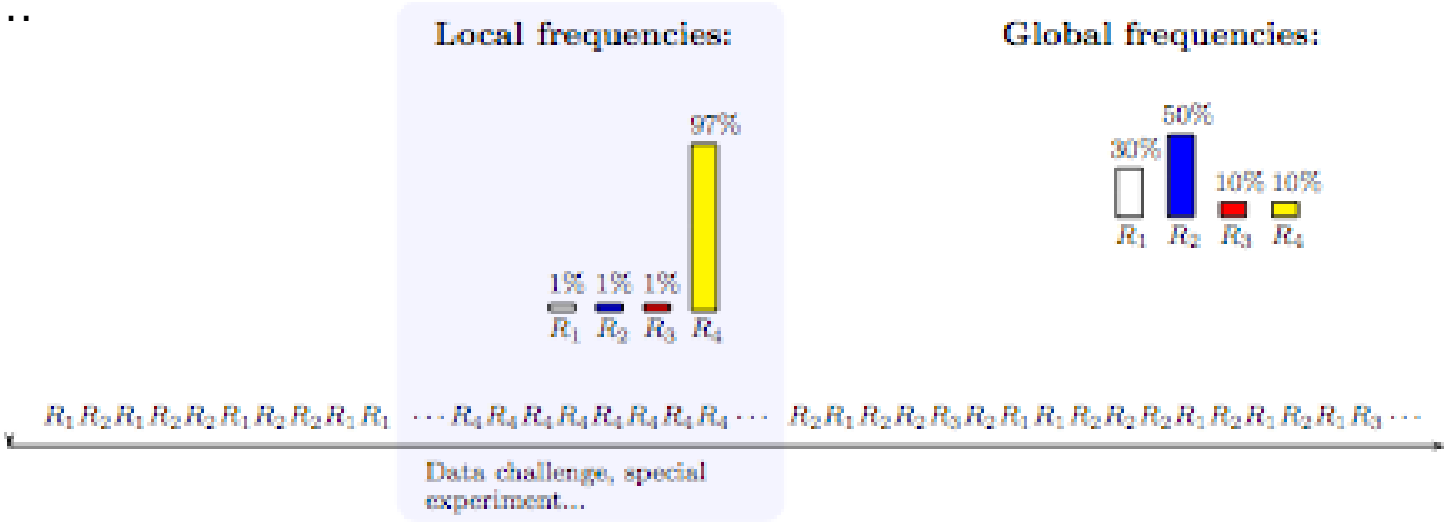


Using SRA



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

SRA with frequency variation detection

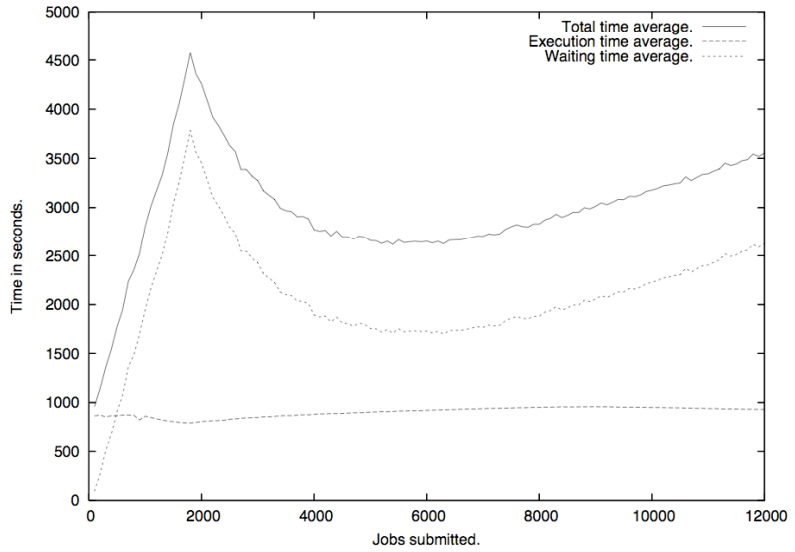
- Algorithm principle

  - SRA gives an initial data distribution
  - The Resource Broker records the submissions and update the frequencies
  - If a frequency varies beyond a ε 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

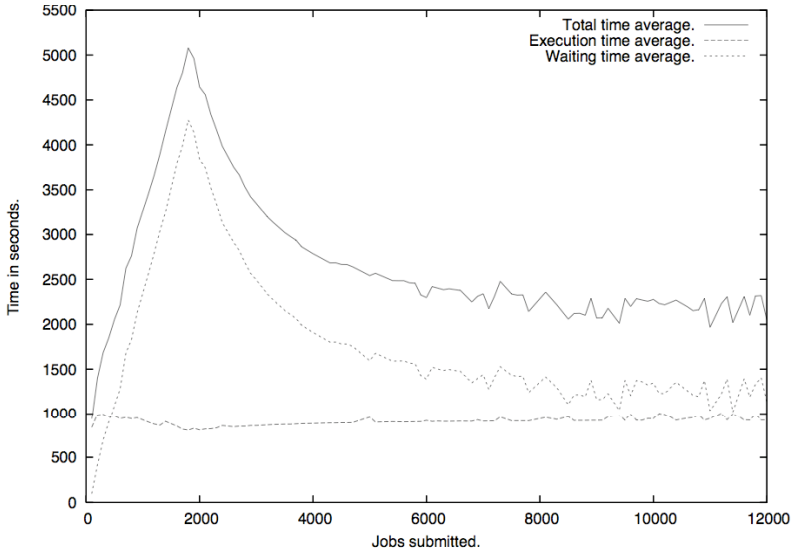
init($f_{k,j}$)
$nb \longleftarrow 0$
$\delta_i^j,\ n_i(k,j) \longleftarrow$ SRA($f_{k,j}$)
While (The scheduler can receive new tasks $T_{k,j}$)
  Record $T_{k,j}$ in $f'_{k,j}$
  If ($nb \geq N$ **AND** $\exists\ k, j$ such $|f_{k,j} - f'_{k,j}| \geq \varepsilon$)
    $\delta_i^j,\ n_i(k,j) \longleftarrow$ SRA($f_{k,j}$)
    Redistribution(δ_i^j) *(if asynchronous transfers are possible)*
    $nb \longleftarrow 0$
    Re-init($f'_{k,j}$)
  EndIf
  Task scheduling using $n_i(k,j)$
  $nb \longleftarrow nb + 1$
EndWhile

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.

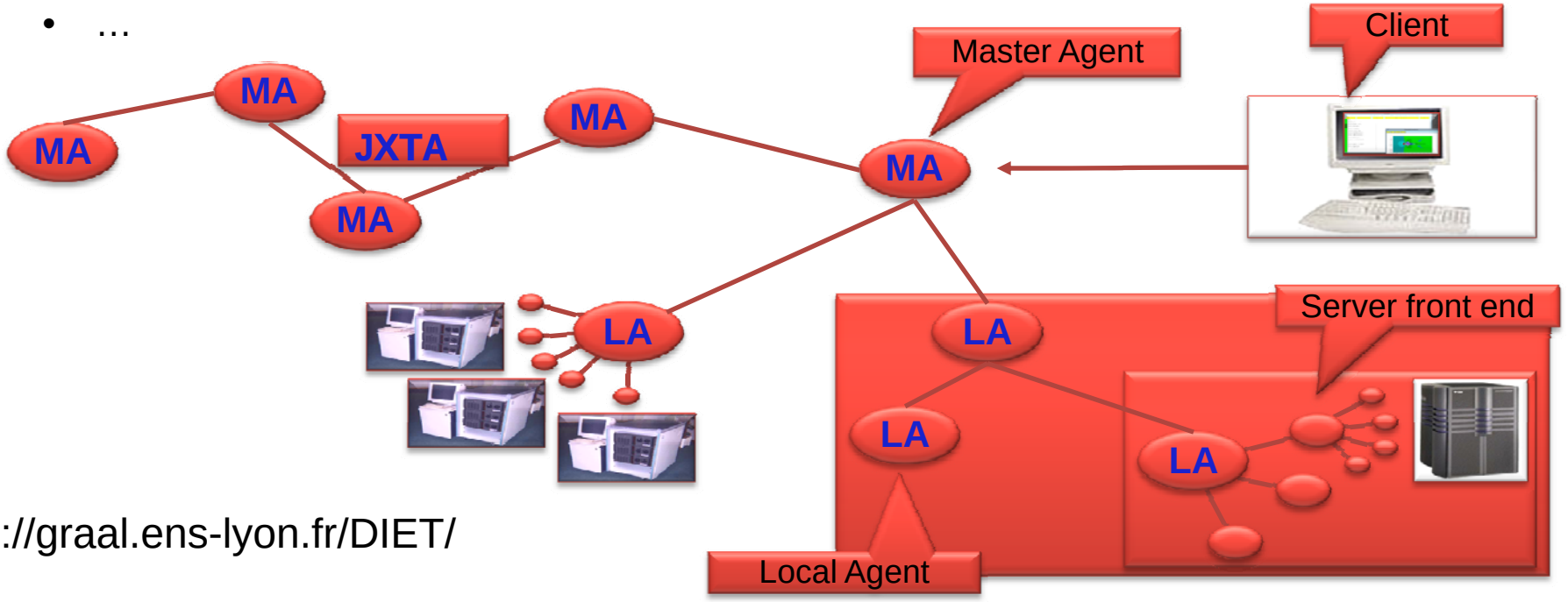


- The frequencies vary - We detect the variations and proceed to the data redistribution
  - The scheduling efficiency is saved.
  - The average execution time of the jobs does not increase too much.



Implementation within a middleware - DIET

- GridRPC API
- Network Enabled Servers paradigm
- Some features
  - Distributed (hierarchical) scheduling
  - Plugin schedulers
  - Data management
  - Workflow management
  - Sequential and parallel task (through batch schedulers)
  - …



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

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)
    - Replication
    - Joining task scheduling and data management
- Work done within the GridRPC Working Group (OGF)
  - Relations with workflow management

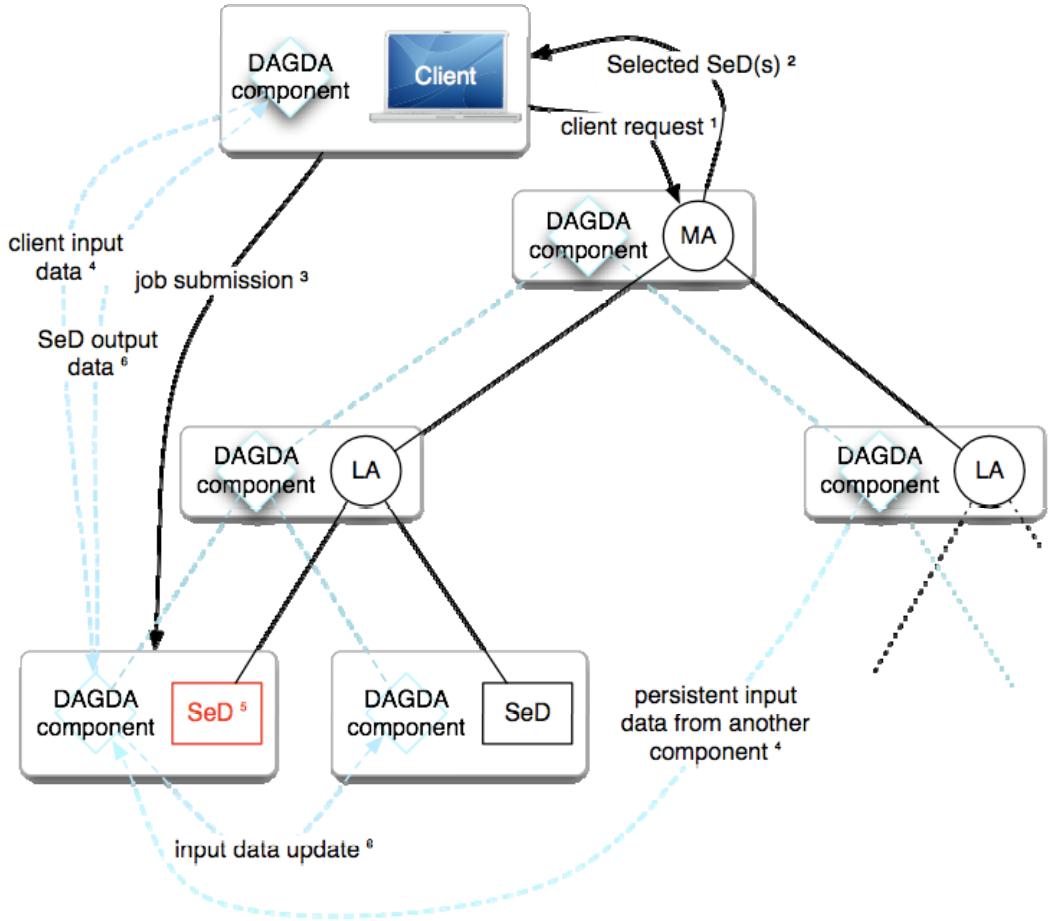
DAGDA

Data **A**rrangement for **G**rid and **D**istributed **A**pplications

- A new data manager for the DIET middleware providing
  - Explicit data replication: using the API
  - Implicit data replication: data items are replicated on the selected servers
  - 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 best source
    - Using statistics on previous transfers
  - Storage resources usage management
    - The space reserved for the data can be configured by the “user”
  - Data status backup/restoration
    - Allowing to stop and restart DIET, saving the data status on each node

DAGDA

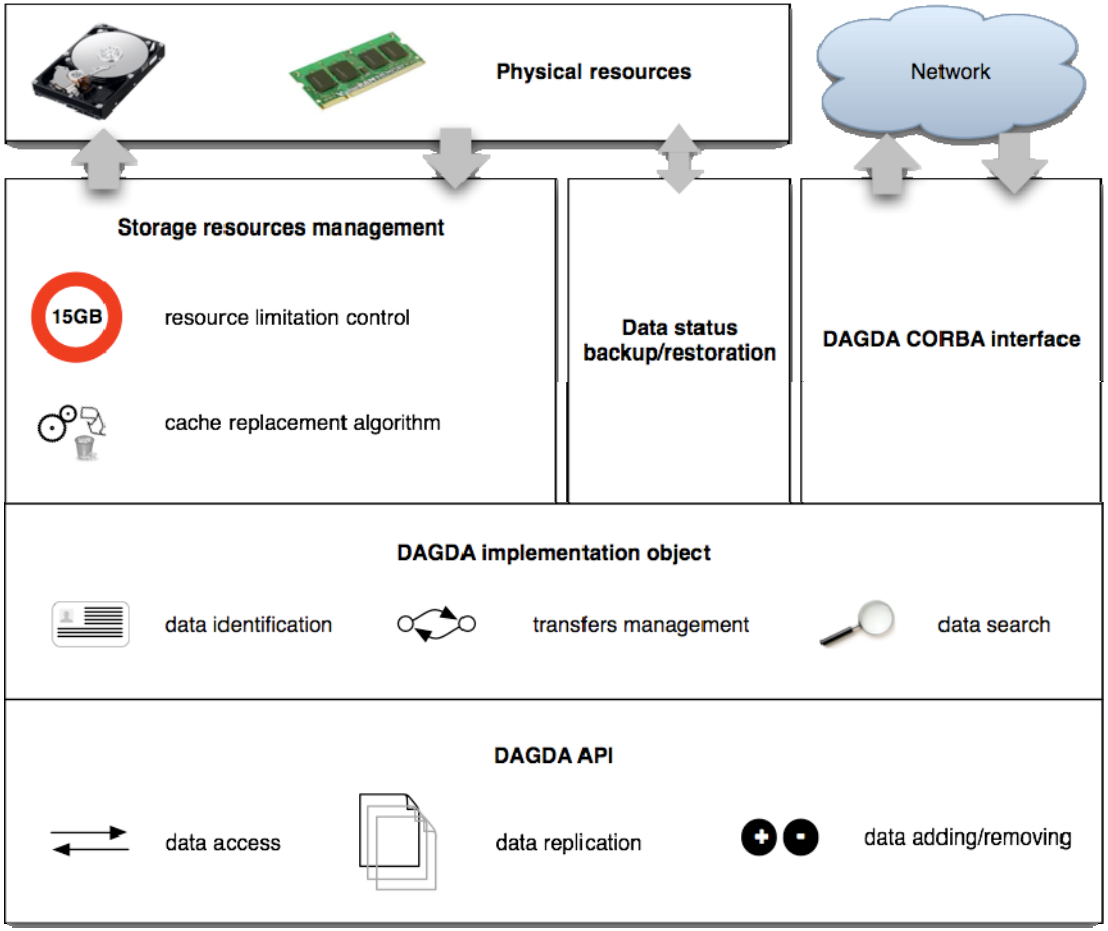
- Transfer model
  - Uses the pull model.
  - The data are sent independently of the service call.
  - The data can be sent in several parts.



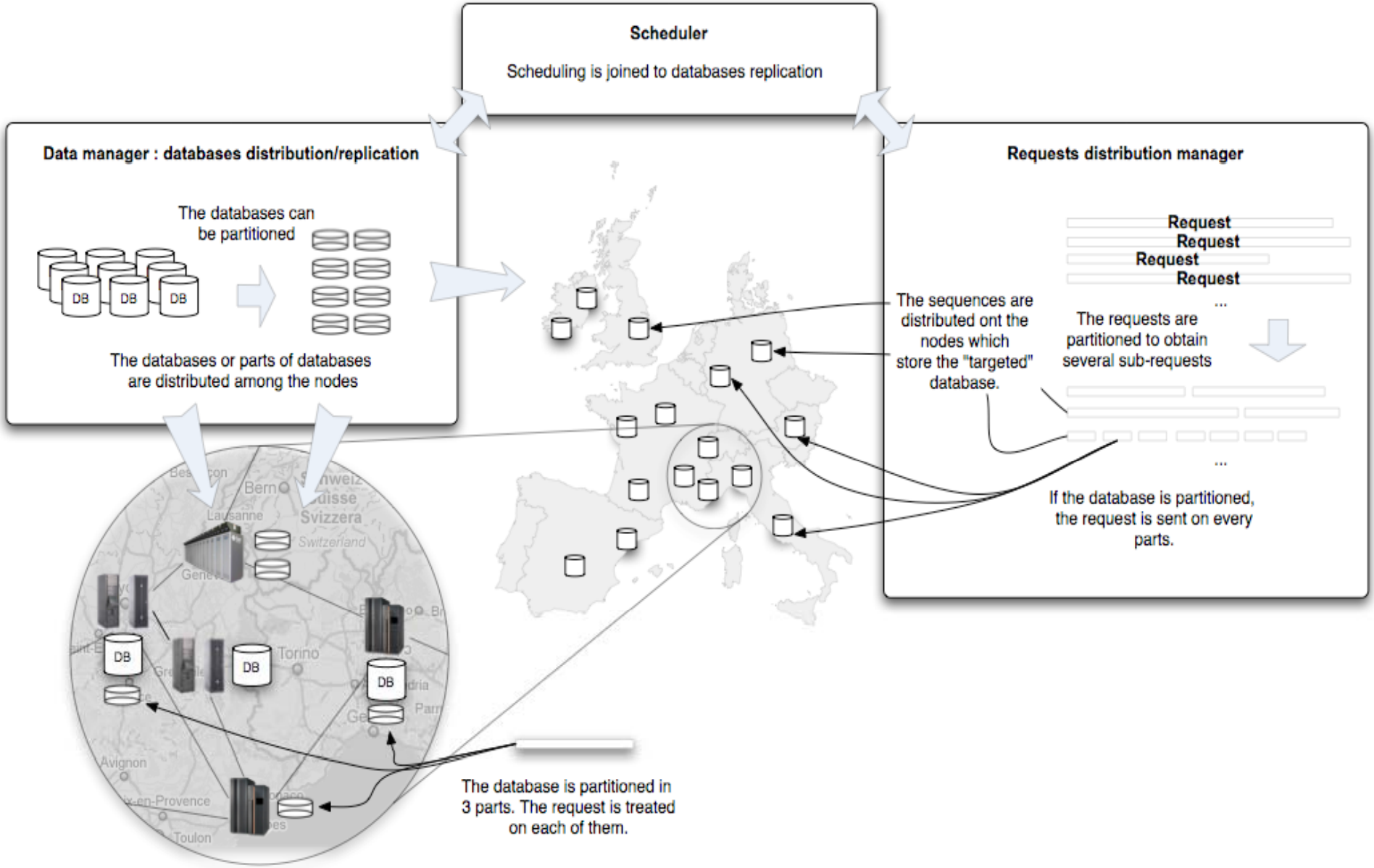
1: The client send a request for a service
2: DIET selects some SeDs according using a scheduling heuristic
3: The client sends its request to the SeD
4: The SeD downloads the data from the client and/or from other DIET servers
5: The SeD performs the call.
6: The persistent data are updated

DAGDA

- DAGDA architecture
  - Each data is associated to one unique identifier
  - DAGDA control the disk and memory space limits. If necessary, it uses a data replacement algorithm
  - The CORBA interface is used to communicate between the DAGDA nodes
  - Users can access data and perform replications using the API



Putting everything together



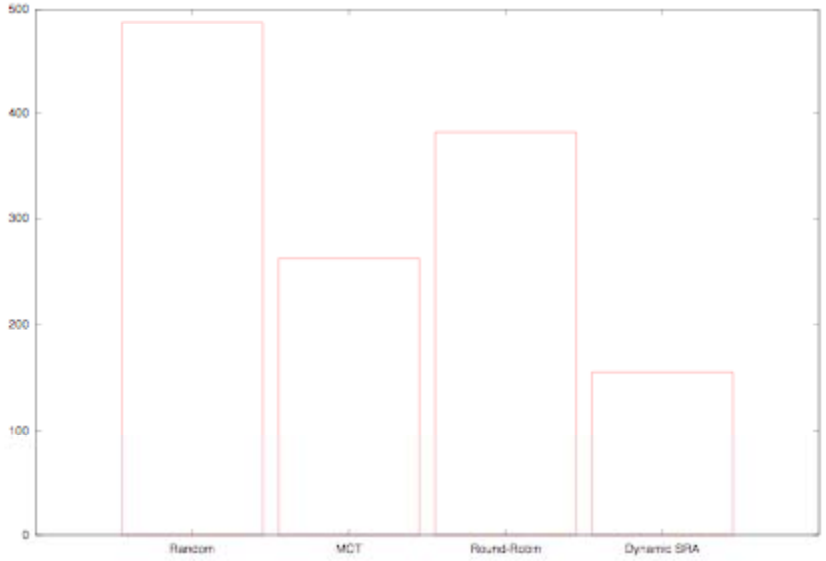
Putting everything together

- Experiments over Grid'5000 using DIET and DAGDA
- 300 nodes (4 sites), 40000 sequences, 4 BLAST algorithms, 2 databases

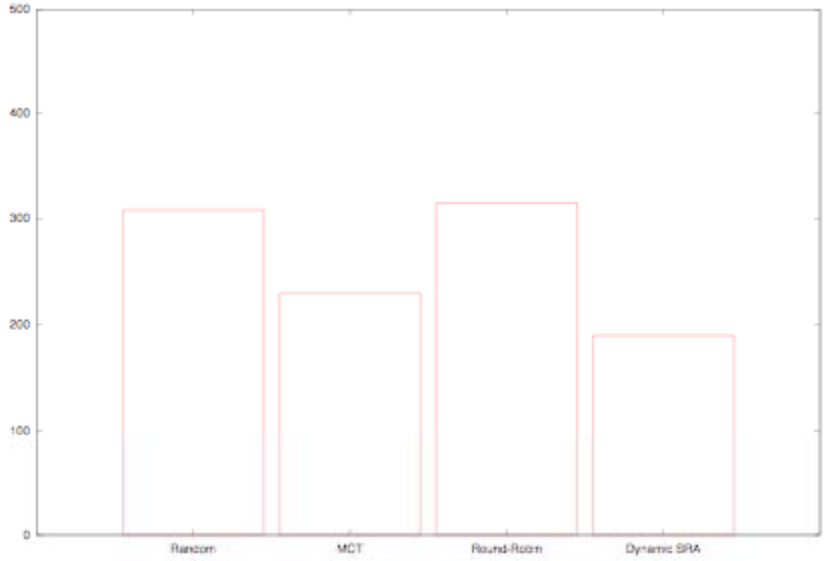
- 2 different sequences splits
  - Small grain: 1 sequence after an other
  - Coarse grain: splitting as a function of the number of servers available
- 4 scheduling algorithms
  - random, MCT, round-robin, dynamic SRA

Putting everything together

- Maximum request partition



- Partition in *n* sub-requests. (*n* the number of available nodes)



- For MCT, Random & Round-Robin: The better requests partitioning is using the number of available nodes (coarse grain).

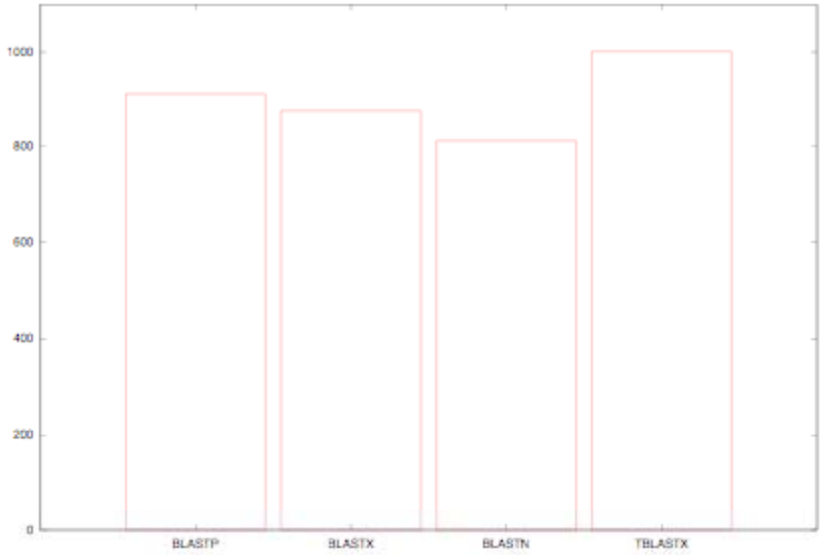
- For Dynamic-SRA: One sequence per request (small grain)

Putting everything together

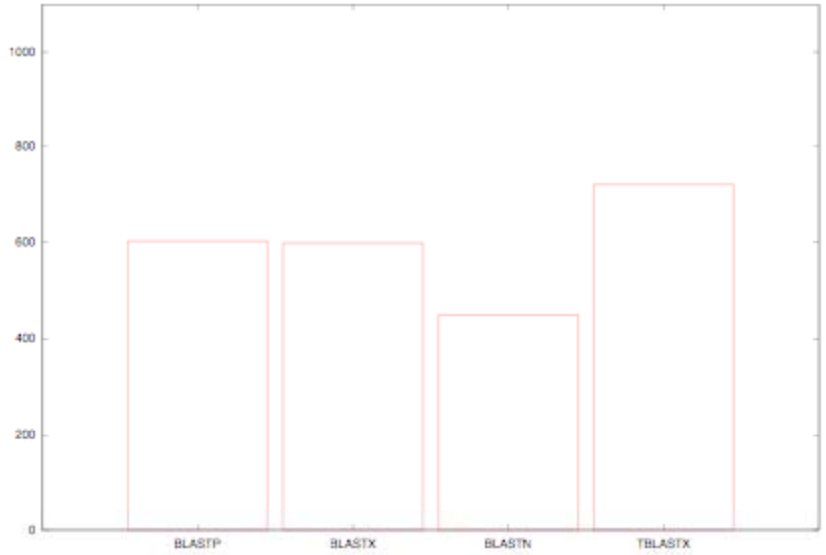
- Comparison of MCT and dynamic SRA over 1000 nodes (8 sites)
- 4 databases between 175 Mo and 6.5 Go
- 4 algorithms
- 40000 sequences
- Frequencies variation

Putting everything together

- Using MCT:



- Using Dynamic-SRA:



- Requests 1 to 20000:
  - BLASTP Vs DB 1 30%
  - BLASTN Vs DB 2 30%
  - BLASTX Vs DB 3 20%
  - TBLASTX Vs DB 4 20%

- Requests 20001 to 40000:
  - BLASTP Vs DB 1 10%
  - BLASTP Vs DB 3 30%
  - BLASTN Vs DB 2 10%
  - BLASTN Vs DB 4 10%
  - BLASTX Vs DB 1 10%
  - BLASTX Vs DB 3 10%
  - TBLASTX Vs DB 2 20%

Conclusion and future work

- Data management is one of the most important issue of large scale applications over the grid
- It has to be linked with request scheduling to get the best performance

- Static algorithms perform well even on a dynamic platform

- **Future work**
  - Provide a set of replication algorithms tuned for specific application classes within DAGDA
  - Increase the exchange between resource and data management systems
  - Use other performance metrics (fairness)
  - Manage replication for workflows scheduling

Questions ?