Data access optimization on grid systems

László Csaba Lőrincz, Tamás Kozsik, Attila Ulbert and Zoltán Horváth
Department of Programming Languages and Compilers
Eötvös Lor´and University
Budapest, Hungary
e-mail: {lesliel|kto|mormota|hz}@inf.elte.hu

***Abstract*— The execution of data intensive grid applications still raises several questions regarding job scheduling, data migration and replication. The optimization techniques applied by these services significantly determine how fast a job can be executed and how early the user can get the execution results.**

In this paper we present strategies for scheduling the execution of data intensive applications. We deem that by taking into account the way applications access their data, the grid middleware can achieve lower response times and earlier execution results. Therefore, we (1) monitor the execution of jobs and gather the necessary resource access information, (2) analyze the compiled information and generate a description of the behavior of the job, and (3) use the generated behavior description to implement optimized scheduling algorithms. This technique can be extremely useful in the case of parameter-sweep applications.

I. Introduction

Grid applications need different types of resources, such as CPU, memory, network bandwidth and secondary storage. The grid middleware tries to allocate these resources to the applications in a way the execution can be as fast and as efficient as possible.

The main focus in this paper is on the secondary storage, on data access. Many applications read and write large data files during their operations. Often these files are not available on the computing element where the application runs. The optimal strategy for accessing these files depends on the kind of the application: computation-intensive applications may spend long periods of time between successive data accesses, while data-intensive applications may need to access large chunks of data very frequently. Consequently either a small buffer may be enough for storing a local copy of the data needed in the next step of computation, or a complete copy of a huge file may be necessary in advance. The access method as well as the granularity of the read/write operations may vary with each input and output file used. Therefore the optimal data access may require the copying of some files as a whole, as well as accessing other files in parts.

In order to optimize the resource allocations and consumption of a job, information about the way it accesses data is required. This information can be specified by the programmer or can be derived from data collected by a monitoring tool. Our approach can support monitoring.

In our conception (see Figure 1) in a GRID system there would be many users, who would run parameter-sweep applications. These applications are executed not only once but several times for similar input values on the same or similar data files. Our goal is to add support to the GRID middleware for collecting data about the execution of such jobs, and to optimize the running process of these applications based on the accumulated information.

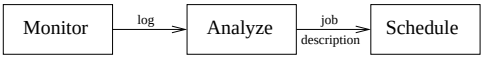

Fig. 1. The Optimization Process

During the first execution of a parameter sweep application (this can be just a test run) a monitoring tool will accumulate the information about the resource usage (and especially about the data access) of the job. The collected data will be processed then by a special tool and the result will be added to the job descriptor file of the application. This includes the extension of the Job Description Language (JDL) [2] and the insertion of the additional data in the current job descriptor files.

Using this extended job descriptor file we can implement new schedulers that would benefit from this new information. Besides choosing the best Computing Element [1] for the next execution of the job, the new scheduler should also be able to send commands to the Replica Manager [1]. These commands would specify the files that must be copied to or in the vicinity of selected Computing Element.

II. Used resources

The Grid Information System (GIS) [1] can store various information that can be used by the Resource Broker through the scheduling process. The middleware provided by the Hungarian ClusterGrid [6] does not provide us all of the required data.

The presently available description of the grid resources contains the type and number of processors in a cluster, the operating system running on it as well as the list of the pre-installed softwares / packages available.

The additional information we would need are:

- The read and write speed of the (local) secondary storage of each grid component.
- The average network bandwidth between two different grid component.
- Besides the number of jobs waiting for execution and the job-queue of a Computing Element (this information is already available) we also require the estimated running time of every single job in the queue as well as the

estimated time the given Computing Element will finish the execution of the current job.

After processing and analyzing the data collected through job execution monitoring, we will acquire some further information about the input and output file usage of the job. We will call this the Data Access Pattern of the application, and it is composed of:

- Type of the file (input or output).
- Which part of the file is used (access_ratio).
- The file usage redundancy (intersection_ratio). This value will describe the average overlapping that exists between the data accessed by a predefined (and configurable) number of consecutive read or write operations on this file.
- A list of the datablock information. Each datablock contains:
  - The file access method (sequential or random) used in the section
  - The starting and ending positions defining the current datablock through four values (two of them representing the absolute and two of them the relative file positions)
  - The distance (step) between starting position of two successive data access operations (in case of sequential file access). This number can be also negative if the job steps backward between these operations.
  - The size (in bytes) of the data processed in a single operation
  - The frequency of the data access operations: the minimum and the average system time (in milliseconds) and CPU time (in mips) between two consecutive operation.

The following example depicts the data access pattern generated for an application that reads file ”test1”:

```
<file_in name="test1" size="100000">
  <sequential>
    <datablock min_pos_absolute="0"
      max_pos_absolute="24000" step="2000"
      size="1000" />
    <timing average_mips="3" />
  </sequential>
  <sequential>
    <datablock min_pos_absolute="25000"
      max_pos_absolute="49000" step="2000"
      size="2000" />
    <timing average_mips="19" />
  </sequential>
  <random>
    <area lower_bound_absolute="50000"
      upper_bound_absolute="100000"
      avg_size="3300" />
    <timing average_mips="2" />
  </random>
</file_in>
```

According to the data access pattern the application processes file ”test1” in the following way:

- In the first part the application reads sequentially blocks of 1000 bytes (skipping the next 1000 bytes)
- In the second part the application reads sequentially blocks of 2000 bytes
- In the third part the application reads the blocks randomly

III. Monitoring

In cases when the application developer is not able to specify the required data access patterns or when the exact behavior and source code of the application is not known, we should use monitoring to gather this information.

The monitoring of jobs can be done in the following ways:

- Altering the source code of the application
- Altering the compiler
- Altering the run-time system
- Altering the operating system.

The software provided by the Hungarian Clustergrid currently is based on the Red Hat Linux 6.2 platform. So in the following the possibilities to merge monitoring code with the code of an application will be discussed focusing on the programming language C (other languages should be handled in a similar way).

From the possibilities listed above, we have chosen to extend the jobs with monitoring code by altering the run-time system by interfering during run-time, when the shared libraries are linked to the application. It is possible to hack the run-time system on a Computing Element node in a way that certain IO function calls are not dispatched to the default IO libraries, but to a custom shared library which performs monitoring as well as executing the actual IO functions.

The advantage of this approach is that the source code of the original application is not needed, since no recompilation is necessary. Only a special shared library must be available on the Computing Element nodes, and an environment variable must be set, in order the data accesses of the job to be monitored or not. This kind of monitoring is completely hidden by the Grid middleware, and is transparent to the scientist who executes the application.

However, this approach is platform-dependent: the techniques presented above can be used on Linux and Solaris systems, porting them to other operating systems require further investigations.

The monitored resources can be divided into two groups:

- CPU and memory,
- Network bandwidth and secondary storage - data access related resources.

Data access monitoring is based on the logging of standard file handling operations defined in the *stdio.h*, *fcntl.h* and *unistd.h* libraries. The information collected for these functions are the name of the operation, the file or stream descriptor, the name of the file and the opening mode flags, the amount of data read or written, or the new position in the stream.

CPU usage monitoring is also bound to the data access. From the optimization point of view it is enough to collect the

CPU usage information between two consecutive file access operations. In order to have a system-independent value we are using the CPU usage percentage and the CPU usage time as a source for this information. These values along with the performance descriptors of the monitoring and the target systems (specified in BogoMips) can be used to estimate the execution time of the given code fragment on a given target system. The performance descriptor (in BogoMips) of the system that is tracking the job execution, will also be stored in the output file of the monitoring (it can be retrieved from the /proc/cpuinfo file).

Monitoring CPU and memory consumption is not possible by overriding some system functions. Instead, the */proc* - process information pseudo-filesystem [5] - can be used for accessing the kernel data structures containing the required information. There is a individual subdirectory for every running process named by the process ID. Every directory contains pseudo-files and directories storing data concerning: the currently mapped memory regions, the status information about the process (state, CPU time, virtual memory size), the free and used memory (both physical and swap).

Tracking of CPU usage is done in a similar way that the Linux program called ”top” works.

In the current implementation the monitoring (the logging of the resource usage) and the analysis of the collected data (the creation of the job descriptor extension) are separated. But in the future it would be possible to merge these actions. Based on the previous measurements the analysis and the generation of the job descriptor extension takes less then 1% of the running time of the whole application.

IV. The analyzer

The data collected by monitoring must be further processed in order to get the desired compact XML description of the data access patterns. The resulting (behavioral) descriptions are used to extend the (static) job descriptions (using an extension of the JDL language), which in turn can be utilized by the grid middleware services to perform the optimization.

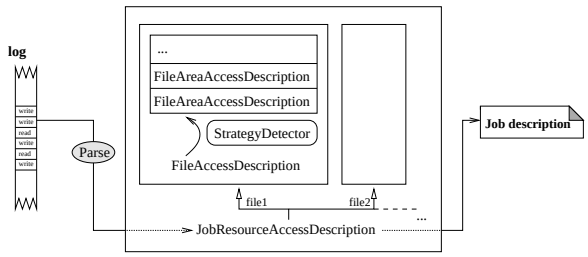

Fig. 2. The Analyzer

The input of our prototype analyzer (see Figure 2) is the log file returned by the monitoring component. Although job monitoring and behavior analyzing are separated processes in our prototype system, the analyzer was designed to allow us to invert the two processes into a united algorithm, which would generate the job behavior description parallel with its execution. Therefore, the analyzer processes the input file sequentially, does not look into the ”future”, and looks into the ”past” only with limitations.

For each file accessed by the job the analyzer builds a *file access description*, which consists of one or more *file area access description(s)*. A *file area access description* describes the file access strategy applied by the job when accessing a specific part of a file. During the processing of the log file, the analyzer continuously keeps track of the file area access strategies used by the job. The analyzer recognizes two kinds of *file area access strategies*: *random* and *sequential* (increasing or decreasing) strategies.

Each kind of strategy is associated with certain behavior characteristics. The strategies are characterized by the following *behavior parameters*:

- the average size of the blocks accessed by the individual file operations,
- the average time elapsed between two subsequent file operations working on the given file,
- the minimum and maximum file position accessed by the job, and the number the job changes these positions.

When the analyzer processes the next log entry, it refines the corresponding *file access description* by either refining the latest *file area access description* of the *file access description* or adding a new *file area access description*. The changes in the applied file access strategy is detected by re-calculating the *behavior parameters* and comparing the new values with the previous ones. If a parameter change were larger than a specified threshold value, the yet actual *file area access description* will be closed and a new one will be added to the *file access description*. For example, if the maximum file position were needed to be updated in the case of a *decreasing sequential* strategy, the analyzer will decide that the job stopped using the *decreasing sequential* strategy, and it will try to determine the new strategy.

The detection of the behavior changes is based on the *access log* the analyzer maintains for each file accessed by the job. An access log entry holds the position and size of the datablock accessed by the job, and the time elapsed since the last file access. The size of the access log(s) is limited allowing the analyzer to detect and determine the file access strategy changes in $O(1)$ time.

In order to determine the new file access strategy, the analyzer resets all behavior characterization parameters and the access log. At this point, the file access strategy is *undetermined*. After the analyzer processes enough file access operations and fills the access log, it determines the new strategy. Please note that the analyzer actually detects changes of file access *behavior*. This means that the *new* strategy is not necessarily a different *kind* of strategy, but a file access strategy having different *behavior parameters*. For example, it the job processes a file sequentially, but from a certain point it will take much more (or less) time to process a data block, the analyzer will decide that the strategy has been changed, and the new strategy is still *sequential increasing*, but its timing characteristics are different.

The file access strategy is determined in the following way:

- the strategy is *increasing sequential* if the maximum position is changed more times than a threshold value (e.g. if the access log size is 10, and the threshold is 7, the maximum position has to be updated 8 times after processing 10 file operations related to the given file)
- the strategy is *decreasing sequential* if the minimum position is changed more times than a threshold value
- otherwise the strategy is *random*

After the strategy is determined, whenever a new file operation is processed, the analyzer updates the access log and the characteristics parameters and checks if the actual file access strategy has changed.

The analyzer algorithm has several parameters, which determine how detailed the resulting *file access description* will be:

- access log size: Specifies how deep the analyzer can look into the past. The larger this parameter the less detailed the description is.
- progress detection threshold: Specifies how many times the maximum (minimum) position has bo be changed in order to detect the increasing (decreasing) sequential access.
- behavior parameter variation: Determines the scale the behavior parameters can change.
- datablock log size: Determines how precise the access and intersection ratio will be. The access and intersection rations are calculated by registering (per file) the past few datablocks accessed by the job.

The goal of the analyzer algorithm is to provide a ”good” description of the job behavior. Unfortunately, currently we cannot give an exact definition of the goodness of descriptions. The description cannot be too detailed that could hide the intrinsical file access behavior, but it cannot also leave out of consideration the real behavior changes. In order to obtain the desired ”good” descriptions we run the algorithm with several actual log files and fine-tune the parameter values.

V. Scheduling strategies

In this section we will present the proposed scheduling strategies, their output, and the possible fields of application. The first one is a basic, prototype strategy, while the second one is a proposal that is not supported yet by the grid middleware, and it would mostly present recommendations for the future grid developments towards optimized scheduling and resource usage and the introduction of grid accounting.

Using the additional information provided by the GIS and the extended job descriptor we can implement new scheduler strategies that besides choosing the most suitable Computing Element for the execution of the job, could also generate file replication commands for the Replica Manager [1]. These commands would specify the files that need to be copied to or in the vicinity of the grid element the job will run on.

During the design phase of the strategy, we have made a few assumptions. The first one is that only one job is running at a given time on a Computing Element (the job is utilizing 100% of the resources available on the Computing Element). The second one is that the jobs will open all of their input and output files at the beginning of their running process and will close them at the end of the execution. We also assume the worst case while estimating times: the input files can be transferred only when all of the preceding jobs have finished their execution.

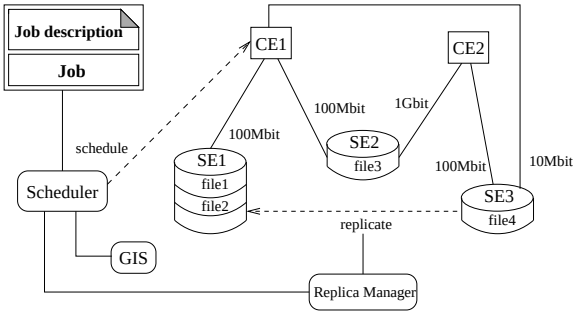

Fig. 3. Scheduling of Jobs

Based on the data access patterns we would predict the processing rate of the input data and the generating rate of the output data (see Figure 3). If the job accesses large chunks of data it is most likely a good idea to schedule it to the Computing Element (or in its neighborhood) where the input files are available. However if the job would have to wait too long before it can be started on the chosen Computing Element, than it would be worth copying the input files to another grid component where the job can be executed earlier. In case of jobs that are less data intensive (use less and smaller input files), the nearness of the files is not so important, as the cost of the replication is very low. If the size of the output files is big, and if they should be copied to a predefined target after the job is finished, than this aspect should be also considered during scheduling process.

A. Static data feeder strategy

Draft: This strategy we will walk through the suitable Computing Elements and will estimate the time the job would finish its execution if it would run on the selected grid component (termination time). The decision is based on the extended job descriptor, and on the information collected from the GIS and the Replica Manager. The output of this strategy is a list of the Computing Elements the job can be run on ordered by the estimated termination time of the job, and optional commands for the Replica Manager that should be executed before the job can be started. In this case the execution of the job is composed from three steps: the input files are copied to the chosen Storage Elements, the job is executed, the output files are copied to the destination specified in the job descriptor.

In the first step, the strategy will select from all of the Computing Elements, those, the job can be run on. The second step is responsible for sorting these Computing Elements, for estimating the termination time of the job on every one of

them and for determining the commands the Replica Manager should execute before the job can be started.

In order to finish the execution of a job on a Computing Element the following steps should be taken:

1) All of the jobs waiting in the queue of the Computing Element should be finished. The duration of the phase can be calculated as the sum of the estimated running time of the jobs.
2) The input files should be copied to the Computing Element. The duration for this can be estimated from the size of the files and the average network bandwidth.
3) The job will be executed. This time can be estimated based on the extended job description, and it is composed from the file access durations and the time of the computation. The first one can be calculated from the amount of data accessed and the speed of the secondary storage, while the second one can be estimated with the help of the monitored CPU time.
4) The output files should be copied to the destination specified by the user. The duration should be calculated similarly to phase two.

Adding the time values calculated above to the (estimated) time the Computing Element can start the job we will obtain the estimated termination time of the job.

The ordered list of Computing Elements will be created using the sum of the time values calculated above and the estimated duration of the commands for the Replica Manager.

This strategy will try to schedule the jobs in a way they would be finished as early as possible.

B. Dynamic data feeder strategy

During the monitoring and the analysis we are collecting more information than the first strategy needs for the optimization. This excess data can be used in this second strategy. The base idea is to download during runtime relevant and sequentially accessed part of the input files and to upload at the same time the output of the job to the specified destination. So we are not dividing the execution of the jobs in three separate phases (download, execution, upload), instead we are trying to execute all of them at the same time: we are not providing the input data statically but dynamically.

The advantage of this approach is that the Computing Element can take different actions during the running process (e.g. download and upload data to and from the local storage). This leads to improved utilization of the grid resources.

The target of this strategy is to predict which part of the input files will be used by the job, and to download only these parts. This can improve the download speed as well as the overall execution time of the job. This strategy should not be confused with the usage of a network file system, that in certain situations can dramatically decrease the performance of a Computing Element. Instead it should be considered a *cache*-ing strategy: the predicted parts of the input files are copied to the local storage of the Computing Element during the job execution, but before the job wants to access this data.

In order to implement this strategy, an improved grid middleware is required. It is not enough any more the collaboration between the scheduler and the Replica Manager; we would need a grid-level file system, that has a cache handler component collaborating with the scheduler. This middleware is not available yet, so the algorithm presented above is only in the proposal phase.

VI. Simulation results

The current version of the Hungarian grid is very limited, due to its under development stage. Therefore we performed simulations to be able to evaluate the efficiency of our approach. We extended OptorSim v2.0 [11] with our static data feeder based scheduler implementation, and allowed to configure the performance of the Computing Elements.

We configured OptorSim to use the EDG topology specified by the configuration file shipped with the simulator. We set different MIPS values of the computers comprising the ‘Imp Coll’, ‘UK’ and ‘Swed’ CEs, while the rest of the CEs had been set to the same MIPS value. The group of jobs the submitted to the Grid was extended with our job: the *gzip* program compressing a specific file. Before we performed the simulation we supplied OptorSim the necessary job description, which was generated after monitoring the execution of *gzip*.

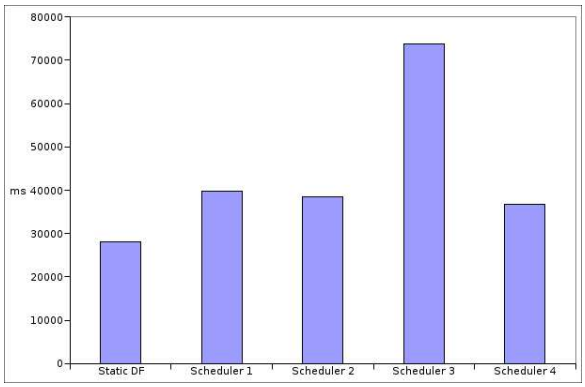

Fig. 4. Mean Job Times

The preliminary simulation results show the major charateristics of or scheduler (see Figure 4). Using our scheduler (Static DF) the mean job time of all jobs on Grid is about 29% lower than in the case we were using the scheduler that considered file access cost and job queue access cost (FAC + JQAC).

The lower mean job time is accompanied by higher effective network usage (ENU), which means that our scheduler requires more file transfers than the other schedulers implemented by OptorSim (see Figure 5). However the dynamic data feeder based scheduler requires less file transfers, therefore the ENU characteristics of the full scheduler and replica manager implementation will be lower. Besides we optimize to the job ending time, and from this point of view the the higher ENU has only limited significance.

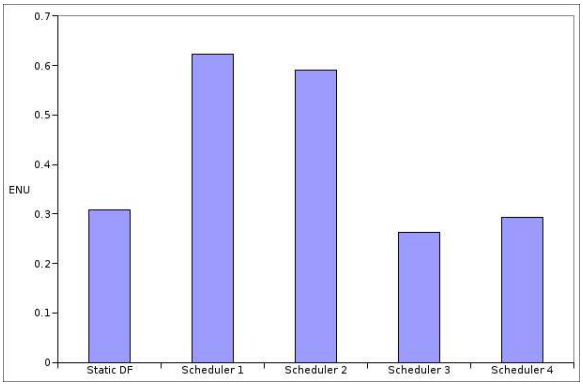

Fig. 5. Effective Network Usage

VII. Future Work

We are planing to improve the generated job descriptor extension by monitoring further the jobs of a parameter-sweep application.

The monitoring and the analysis of the collected data should also be merged into a single action. This would avoid the creation of the (large) log files.

Similarly to the generation of data access patterns we would like to create communication patterns for applications composed of parallel processes (e.g. PVM tasks).

In order to get a more detailed view of the efficiency of our architecture, we will perform more thorough simulations inlcuding wide range of grid topologies and jobs. As part of the simulation we are planning to extend OptorSim.

We are also planning to transfer the solutions presented in this paper from the currently used EDG middleware to the one used by EGEE [7] in the LCG-2 project [8], and then also to members of the next generation of grid middleware, like gLite [9].

VIII. Related work

Many different scheduling and data replication algorithms exist, and many can be adapted on GRID systems. Some of them opt for simplicity, others—like [12]—make use of more sophisticated decision algorithms. Our approach belongs to the second category. A good summary about relevant research can be found in [13]. In contrast to [13], we prefer centralized scheduling decisions. We feel that such decisions can make better use of the information collected about the resources of the grid system. One approach that is probably the closest to ours is introduced in [14]. It also addresses scheduling parameter sweep applications; it focuses on the efficient co-location of data and programs using them, and on adaptive scheduling. The novelty of our approach is to build a middleware component that would be able to move data in pieces to the programs that request them, thus making the replication of whole data files unnecessary.

IX. Conclusion

We have presented a new approach in job scheduling optimization on grid systems. Our goal is to optimize the ending (and not the execution) time of the job. The basic idea is to improve mainly the data accessing performance. This can be done either by choosing the correct node for running the job complemented with possible file replications or by dynamically downloading and uploading only the processed data during runtime.

Both solutions depend heavily on the way the job access its input and output files.

The additional information required for the scheduler can be specified by the developer of the job or can be collected through monitoring and then processed by an analyzer tool. The second case solution be used especially in the case of parameter-sweep applications.

Currently, the presented architecture is being implemented on the Hungarian Clustergrid. However, due to its early stage, the efficiency of the designed scheduling strategies are exclusively evaulated through simulations. Although the first results are promising, we need to perform more simulations with different Grid topologies and Job characteristics to get a detailed picture of the designed strategies.

Acknowledgment

This work was supported by IKTA 64/2003 and the Bolyai Research Fellowship.

References

[1] Foster, I.: The Grid: Blueprint for a New Computing Infrastructure. July 1998, Morgan-Kaufmann.
[2] Job Description Language Attributes. http://auger.jlab.org/jdl/PPDG_JDL.htm
[3] Szalai, F.: ClusterGrid Br´oker rendszer kiterjeszthető erőforr´as ¨utemező´enek specifi k´aci´oja. (in Hungarian) http://www.clustergrid.niif.hu/project_en/docs.html
[4] Condor project: Classifi ed Advertisements. http://www.cs.wisc.edu/condor/classad/
[5] LinuxForum: Linux Filesystem Hierarchy, 1.10. /proc. http://www.linuxforum.com/linux-fi lesystem/proc.html
[6] NIIF Supercomputing Center: The Hungarian ClusterGrid Infrastructure Project. http://www.clustergrid.niif.hu/
[7] EGEE: Enabling Grids for E-sience. http://public.eu-egee.org/
[8] LHC Computing Grid Project. http://lcg.web.cern.ch/LCG/
[9] gLite: Lightweight Middleware for Grid Computing. http://glite.web.cern.ch/glite/
[10] M´aray, T., Stef´an, P., Szalai, F., Vit´ez, G.: The Hungarian ClusterGrid Project: Challanges of a Production Grid. http://www.clustergrid.niif.hu/project_en/docs.html
[11] Simulating data access optimization algorithms - OptorSim. http://edg-wp2.web.cern.ch/edg-wp2/optimization/optorsim.html
[12] William H. Bell, David G. Cameron, Ruben Carvajal-Schiaffi no, A. Paul Millar, Kurt Stockinger, Floriano Zini: Evaluation of an Economy-Based File Replication Strategy for a Data Grid. In: International Workshop on Agent based Cluster and Grid Computing at CCGrid 2003, Tokyo, Japan, May 2003. IEEE Computer Society Press.
[13] Kaviatha Ranganathan, Ian Foster: Computation Scheduling and Data Replication Algorithms for Data Grids. In: 'Grid Resource Management: State of the Art and Future Trends', J. Nabrzyski, J. Schopf, and J. Weglarz, eds. Kluwer Academic Publishers, 2003.
[14] H. Casanova, G. Obertelli, F. Berman, R. Wolski: The AppLeS parameter sweep template: User-level middleware for the grid. In: Proceedings of Supercomputing'00, Denver, 2000.