JOB AND APPLICATION-LEVEL SCHEDULING IN DISTRIBUTED COMPUTING

Victor V. Toporkov
Computer Science Department, Moscow Power Engineering Institute, ul. Krasnokazarmennaya 14, Moscow, 111250 Russia
ToporkovVV@mpei.ru

ABSTRACT
This paper presents an integrated approach for scheduling in distributed computing
with strategies as sets of job supporting schedules generated by a critical works
method. The strategies are implemented using a combination of job-flow and
application-level techniques of scheduling within virtual organizations of Grid.
Applications are regarded as compound jobs with a complex structure containing
several tasks co-allocated to processor nodes. The choice of the specific schedule
depends on the load level of the resource dynamics and is formed as a resource
request, which is sent to a local batch-job management system. We propose
scheduling framework and compare diverse types of scheduling strategies using
simulation studies.

Keywords: distributed computing, scheduling, application level, job flow,
metascheduler, strategy, supporting schedules, task, critical work.

1 INTRODUCTION
The fact that a distributed computational
environment is heterogeneous and dynamic along
with the autonomy of processor nodes makes it much
more difficult to manage and assign resources for job
execution at the required quality level [1].

When constructing a computing environment
based on the available resources, e.g. in the model
which is used in X-Com system [2], one normally
does not create a set of rules for resource allocation
as opposed to constructing clusters or Grid-based
virtual organizations. This reminds of some
techniques, implemented in Condor project [3, 4].
Non-clustered Grid resource computing
environments are using similar approach. For
example, @Home projects which are based on
BOINC system realize cycle stealing, i.e. either idle
computers or idle cycles of a specific computer.

Another still similar approach is related to the
management of distributed computing based on
resource broker assignment [5-11]. Besides Condor
project [3, 4], one can also mention several
application-level scheduling projects: AppleLeS [6],
APST [7], Legion [8], DRM [9], Condor-G [10], and

It is known, that scheduling jobs with
independent brokers, or application-level scheduling,
allows adapting resource usage and optimizing a
schedule for the specific job, for example, decreasing
its completion time. Such approaches are important,
because they take into account details of job structure
and users resource load preferences [5]. However,
when independent users apply totally different
criteria for application optimization along with job-
flow competition, it can degrade resource usage and
integral performance, e.g. system throughput,
processor nodes load balance, and job completion
time.

Alternative way of scheduling in distributed
computing based on virtual organizations includes a
set of specific rules for resource use and assignment
that regulates mutual relations between users and
resource owners [1]. In this case only job-flow level
scheduling and allocation efficiency can be
increased. Grid-dispatchers [12] or metaschedulers
are acting as managing centres like in the GrADS
project [13]. However, joint computing nature of
virtual organizations creates a number of serious
challenges. Under such conditions, when different
applications are not isolated, it is difficult to achieve
desirable resource performance: execution of the
user’s processes can cause unpredictable impact on
other neighbouring processes execution time.
Therefore, there are researches that pay attention to
the creation of virtual machine based virtual Grid
The above-mentioned works are related to either job-flow scheduling problems or application-level scheduling.

Fundamental difference between them and the approach described is that the resultant dispatching strategies are based on the integration of job-flows management methods and compound job scheduling methods on processor nodes. It allows increasing the quality of service for the jobs and distributed environment resource usage efficiency.

It is considered, that the job can be compound (multiprocessor) and the tasks, included in the job, are heterogeneous in terms of computation volume and resource need. In order to complete the job, one should co-allocate the tasks to different nodes. Each task is executed on a single node and it is supposed, that the local management system interprets it as a job accompanied by a resource request.

On one hand, the structure of the job is usually not taken into account. The rare exception is the Maui cluster scheduler [16], which allows for a single job to contain several parallel, but homogeneous (in terms of resource requirements) tasks. On the other hand, there are several resource-query languages. Thus, JDL from WLMS (http://edms.cern.ch) defines alternatives and preferences when making resource query, ClassAds extensions in Condor-G [10] allows forming resource-queries for dependant jobs. The execution of compound jobs is also supported by WLMS scheduling system of gLite platform (http://www.glite.org), though the resource requirements of specific components are not taken into account.

What sets our work apart from other scheduling research is that we consider coordinated application-level and job-flow management as a fundamental part of the effective scheduling strategy within the virtual organization.

Environment state of distribution, dynamics of its configuration, user’s and owner’s preferences cause the need of building multifactor and multicriteria job managing strategies [17-20]. Availability of heterogeneous resources, data replication policies [12, 21, 22] and multiprocessor job structure for efficient co-allocation between several processor nodes should be taken into account.

In this work, the multicriteria strategy is regarded as a set of supporting schedules in order to cover possible events related to resource availability.

The outline of the paper is as follows.

In section 2, we provide details of application-level and job-flow scheduling with a critical works method and strategies as sets of possible supporting schedules.

Section 3 presents a framework for integrated job-flow and application-level scheduling.

Simulation studies of coordinated scheduling techniques and results are discussed in Section 4.

We conclude and point to future directions in Section 5.

2 APPLICATION-LEVEL AND JOB-FLOW SCHEDULING STRATEGIES

2.1 Application-Level Scheduling Strategy

The application-level scheduling strategy is a set of possible resource allocation and supporting schedules (distributions) for all N tasks in the job [18]:

\[
\text{Distribution:=} \quad <\langle \text{Task 1/Allocation i}, \{\text{Start 1, End 1}\}, \ldots, \langle \text{Task N/Allocation j}, \{\text{Start N, End N}\}\rangle, \ldots>,
\]

where Allocation i, j is the processor node i, j for Task 1,N,Start 1, N, End 1,N – run time and stop time for Task 1,N execution.

Time interval \{Start, End\} is treated as so called walltime (WT), defined at the resource reservation time [15] in the local batch-job management system.

Figure 1 shows some examples of job graphs in strategies with different degrees of distribution, task details, and data replication policies [19]. The first type strategy S1 allows scheduling with fine-grain computations and multiple data replicas, the second type strategy S2 is one with fine-grain computations and a bounded number of data replicas, and the third type S3 implies coarse-grain computations and constrained data replication. The vertices P1,...,P6, P23, and P45 correspond to tasks, while D1,...,D8, D12, D36, and D78 correspond to data transmissions. The transition from graph G1 to graphs G2 and G3 is performed through lumping of tasks and reducing of the parallelism level.

The job graph is parameterized by prior estimates of the duration Ti of execution of a task Pi for a
processor node \( n_j \) of the type \( j \), of relative volumes \( V_{ij} \) of computations on a processor (CPU) of the type \( j \), etc. (Table 1).

It is to mention, such estimations are also necessary in several methods of priority scheduling including backfilling in Maui cluster scheduler.

The processor node load level \( L_{Lj} \) is the ratio of the total time of usage of the node of the type \( j \) to the job run time. Schedules in Fig. 2, b and Fig. 2, c are related to strategies \( S_2 \) and \( S_3 \).

2.2 Critical Works Method

Strategies are generated with a critical works method [20].

The gist of the method is a multiphase procedure. The first step of any phase is scheduling of a critical work – the longest (in terms of estimated execution time \( T_{ij} \) for task \( F_i \)) chain of unassigned tasks along with the best combination of available resources. The second step is resolving collisions caused by conflicts between tasks of different critical works competing for the same resource.

Figure 2 shows fragments of strategies of types \( S_1, S_2, \) and \( S_3 \) for jobs in Fig. 1.

The duration of all data transmissions is equal to one unit of time for \( G_1 \), while the transmissions \( D_{12} \) and \( D_{78} \) require two units of time and the transmission \( D_{36} \) requires four units of time for \( G_2 \) and \( G_3 \).

We assume that the lumping of tasks is characterized by summing of the values of corresponding parameters of constituent subtasks (see Table 1).

Table 1: User's task estimations.

<table>
<thead>
<tr>
<th>( T_{ij} ), ( Vij )</th>
<th>( F_1 )</th>
<th>( F_2 )</th>
<th>( F_3 )</th>
<th>( F_4 )</th>
<th>( F_5 )</th>
<th>( F_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_{i1} )</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>( T_{i2} )</td>
<td>4</td>
<td>6</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>( T_{i3} )</td>
<td>6</td>
<td>9</td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>( T_{i4} )</td>
<td>8</td>
<td>12</td>
<td>4</td>
<td>8</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>( Vij )</td>
<td>20</td>
<td>30</td>
<td>10</td>
<td>20</td>
<td>10</td>
<td>20</td>
</tr>
</tbody>
</table>

Supporting schedules in Fig. 2, a present a subset of a Pareto-optimal strategy of the type \( S_1 \) for tasks \( F_i, i=1, ... , 6 \) in \( G_1 \).

The Pareto relation is generated by a vector of criteria \( CF, L_{Lj}, j=1, ... , 4 \).

A job execution cost-function \( CF \) is equal to the sum of \( V_{ij}/T_i \), where \( T_i \) is the real load time of processor node \( j \) by task \( F_i \) rounded to nearest not-smaller integer. Obviously, actual solving time \( T_i \) for a task can be different from user estimation \( T_{ij} \) (see Table 1).

The fragments of scheduling strategies \( S_1 \) (a), \( S_2 \) (b), \( S_3 \) (c).

For example, there are four critical works \( 12, 11, 10, \) and \( 9 \) time units long (including data transfer time) on fastest processor nodes of the type 1 for the job graph \( G_1 \) in Fig. 1, a (see Table 1):

\((F_1, F_2, F_4, F_6), (F_1, F_2, F_5, F_6), (F_1, F_3, F_4, F_6), (F_1, F_3, F_5, F_6)\).

The schedule with \( CF=37 \) has a collision (see Fig. 2, a), which occurred due to simultaneous attempts of
2.3 Examples of Scheduling Strategies

Let us assume that we need to construct a conditionally optimal strategy of the distribution of processors according to the main scheme of the critical works method from [20] for a job represented by the information graph $G_j$ (see Fig. 1). Prior estimates for the duration $T_{ij}$ of processing tasks $P_1, \ldots, P_6$ and relative computing volumes $V_{ij}$ for four types of processors are shown in Table 1, where $i = 1, \ldots, 6; j = 1, \ldots, 4$. The number of processors of each type is equal to 1. The duration of all data exchanges $D_1, \ldots, D_8$ is equal to one unit of time. The walltime is given to be $WT = 20$. The criterion of resource-use efficiency is a cost function $CF$. We take a prior estimate for the duration $T_{ij}$ that is the nearest to the limit time $T_i$ for the execution of task $P_i$ on a processor of type $j$, which determines the type $j$ of the processor used.

The conflicts between competing tasks are resolved through unused processors, which, being used as resources, are accompanied with a minimum value of the penalty cost function that is equal to the sum of $V_{ij}/T_{ij}$ (see Table 1) for competing tasks.

It is required to construct a strategy that is conditionally minimal in terms of the cost function $CF$ for the upper and lower boundaries of the maximum range for the duration $T_{ij}$ of the execution of each task $P_i$ (see Table 1). It is a modification of the strategy $S_1$ with fine-grain simulations, active data replication policy, and the best- and worst execution time estimations.

The strategy with a conditional minimum with respect to $CF$ is shown in Table 2 by schedules 1, 2, and 3 ($A_i$ is allocation of task $P_i, i = 1, \ldots, 6$) and the scheduling diagrams are demonstrated in Fig. 2, a.

The strategies that are conditionally maximal with respect to criteria $LL_1, LL_2, LL_3,$ and $LL_4$ are given in Table 2 by the cases 4-7; 8, 9; 10, 11; and 12-14, respectively. Since there are no conditional branches in the job graph (see Fig. 1), $LL_j$ is the ratio of the total time of usage of a processor of type $j$ to the walltime $WT$ of the job completion.

The Pareto-optimal strategy involves all schedules in Table 2. The schedules 2, 5, and 13 have resolved collisions between tasks $P_4$ and $P_5$.

Let us assume that the load of processors is such that the tasks $P_1, P_2,$ and $P_3$ can be assigned with no more than three units of time on the first and third processors (see Table 2). The metascheduler runs through the set of supporting schedules and chooses a concrete variant of resource distribution that depends on the actual load of processor nodes.

### Table 2: The strategy of the type MS1.

<table>
<thead>
<tr>
<th>Schedule</th>
<th>Duration</th>
<th>Allocation</th>
<th>Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$T_1$</td>
<td>$T_2$</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>14</td>
<td>10</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
Then, the metascheduler should choose the schedules 1, 2, 4, 5, 12, and 13 as possible variants of resource distribution. However, the concrete schedule should be formulated as a resource request and implemented by the system of batch processing subject to the state of all four processors and possible runtimes of tasks P4, P5, and P6 (see Table 2).

Suppose that we need to generate a Pareto-optimal strategy for the job graph G2 (see Fig. 1) in the whole range of the duration Ti of each task P1, while the step of change is taken to be no less than the lower boundary of the range for the most performance processor.

The Pareto relation is generated by the vector of criteria CF, LL1, ..., LL4. The remaining initial conditions are the same as in the previous example.

The strategies that are conditionally optimal with respect to the criteria CF, LL1, LL2, LL3, and LL4 are presented in Table 3 by the schedules 1-4, 5-12, 13-17, 18-25, and 26-33, respectively. The Pareto-optimal strategy does not include the schedules 2, 5, 12, 14, 16, 17, 22, and 30.

Let us consider the generation of a strategy for the job represented structurally by the graph G3 in Fig. 1 and by summing of the values of the parameters given in Table 1 for tasks P2, P3 and P4, P5.

As a result of the resource distribution for the model G3, the tasks P1, P23, P45, and P6 turn out to be assigned to one and the same processor of the first type. Consequently, the costs of data exchanges D12, D36, and D78 can be excluded. Because there can be no conflicts in this case between processing tasks (see Fig. 1), the scheduling obtained before the exclusion of exchange procedures can be revised.

### Table 3: The strategy of the type S2.

<table>
<thead>
<tr>
<th>Schedule</th>
<th>Duration</th>
<th>Allocation</th>
<th>Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>T2</td>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>T3</td>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>T4</td>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>T5</td>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>T6</td>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>
Table 4: The strategy of the type S3.

<table>
<thead>
<tr>
<th>Schedule</th>
<th>Duration</th>
<th>Allocation</th>
<th>Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>T23</td>
<td>T45</td>
<td>T6</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>11</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

The results of distribution of processors are presented in Table 4 (A23, A45 are allocations, and T23, T45 are run times for tasks P23 and P45). Schedules 1-6 in Table 4 correspond to the strategy that is conditionally minimal with respect to CF with LL1 = 1. Consequently, there is no sense in generating conditionally maximal schedules with respect to criteria LL1, …, LL4.

2.4 Coordinated Scheduling with the Critical Works Method

The critical works method was developed for application-level scheduling [19, 20]. However, it can be further refined to build multifactor and multicriteria strategies for job-flow distribution in virtual organizations. This method is based on dynamic programming and therefore uses some integral characteristics, for example total resource usage cost for the tasks that compose the job. However the method of critical works can be referred to the priority scheduling class. There is no conflict between these two facts, because the method is dedicated for task co-allocation of compound jobs.

Let us consider a simple example. Fig. 3 represents two jobs with walltimes WT1 = 110 and WT2 = 140 that are submitted to the distributed environment with 8 CPUs. If the jobs are submitted one-by-one the metascheduler (Section 3) will also schedule them one-by-one and will guarantee that every job will be scheduled within the defined time interval and in most efficient way in terms of a selected cost function and maximize average load balance of CPUs on a single job scale (Fig. 4). Job-flow execution will be finished at WT3 = 250. This is an example of application-level scheduling and no integral job-flow characteristics are optimized in this case.

To combine application-level scheduling and job-flow scheduling and to fully exploit the advantages of the approach proposed, one can submit both jobs simultaneously or store them in buffer and execute the scheduling for all jobs in the buffer after a certain amount of time (buffer time). If the metascheduler gets more than one job to schedule it runs the developed mechanisms that optimize the whole job-flow (two jobs in this example). In that case the metascheduler will still try to find an optimal schedule for each single job as described above and, at the same time, it will try to find the most optimal job assignment so that the average load of CPUs will be maximized on a job-flow scale.

Fig. 5 shows, that both jobs are executed within WT4 = WT2 = 140, every data dependency is taken into account (e.g. for the second job: task P2 is executed only after tasks P0, P4, and P1 are ready), the final schedule is chosen from the generated strategy with the lowest cost function.

Priority scheduling based on queues is not an efficient way of multiprocessor jobs co-allocating, in our opinion. Besides, there are several well-known side effects of this approach in the cluster systems such as LL, NQE, LSF, PBS and others.
For example, traditional First-Come-First-Serve (FCFS) strategy leads to idle standing of the resources. Another strategy, which involves job ranking according to the specific properties, such as computational complexity, for example Least-Work-First (LWF), leads to a severe resource fragmentation and often makes it impossible to execute some jobs due to the absence of free resources. In distributed environments these effects can lead to unpredictable job execution time and thereby to unsatisfactory quality of service.

In order to avoid it many projects have components that make schedules, which are supported by preliminary resource reservation mechanisms [15, 16].

One to mention is Maui cluster scheduler, where backfilling algorithm is implemented. Remote Grid resource reservation mechanism is also supported in GARA, Ursala and Silver projects [16]. Here, only one variant of the final schedule is built and it can become irrelevant because of changes in the local job-queue, transporting delays etc. The strategy is some kind of preparation of possible activities in distributed computing based on supporting schedules (see Fig. 2, Tables 2, 3 and 4) and reactions to the events connected with resource assignment and advance reservations [15, 16]. The more factors considered as formalized criteria are taken into account in strategy generation, the more complete is the strategy in the sense of coverage of possible events [18, 19]. The choice of the supporting schedule [20] depends on the utilization state of processor nodes, data storage and relocation policies specific to the environment, structure of the jobs themselves and user estimations of completion time and resource requirements.

It is important to mention that users can submit jobs without information about the task execution order as required by existing schedulers like Maui cluster scheduler were only queues are supported. Implemented mechanisms of our approach support a complex structure for the job, which is represented as a directed graph, so user should only provide data dependencies between tasks (i.e. the structure of the job). The metascheduler will generate the schedules to satisfy their needs by providing optimal plans for jobs (application-level scheduling) and the needs for the resource owners by optimizing the defined characteristics of the job-flow for the distributed system (job-flow scheduling).

3 METASCHEDULING FRAMEWORK

In order to implement the effective scheduling and allocation to heterogeneous resources, it is very important to group user jobs into flows according to the strategy type selected and to coordinate job-flow and application-level scheduling. A hierarchical structure (Fig. 6) composed of a job-flow metascheduler and subsidiary job managers, which are cooperating with local batch-job management systems, is a core part of a scheduling framework proposed in this paper. It is assumed that the specific supporting schedule is realized and the actual allocation of resources is performed by the system of batch processing of jobs. This schedule is implemented on the basis of a user resource request with a requirement to the types and characteristics of resources (memory and processors) and to the system software as well as generated, for example, by the script of the job entry instruction qsub. Therefore, the formation and support of scheduling...
strategies should be conducted by the metascheduler, an intermediary link between the job flow and the system of batch processing.

![Diagram of metascheduling framework]

**Figure 6:** Components of metascheduling framework.

The advantages of hierarchically organized resources managers are obvious, e.g., the hierarchical job-queue-control model is used in the GrADS metascheduler [13] and X-Com system [2]. Hierarchy of intermediate servers allows decreasing idle time for the processor nodes, which can be inflicted by transport delays or by unavailability of the managing server while it is dealing with the other processor nodes. Tree-view manager structure in the network environment of distributed computing allows avoiding deadlocks when accessing resources. Another important aspect of computing in heterogeneous environments is that processor nodes with the similar architecture, contents, administrating policy are grouped together under the job manager control.

Users submit jobs to the metascheduler (see Fig. 6) which distributes job-flows between processor node domains according to the selected scheduling and resource co-allocation strategy $S_i$, $S_j$ or $S_k$. It does not mean, that these flows cannot “intersect” each other on nodes. The special reallocation mechanism is provided. It is executed on the higher-level manager or on the metascheduler-level. Job managers are supporting and updating strategies based on cooperation with local managers and simulation approach for job execution on processor nodes. Innovation of our approach consists in mechanisms of dynamic job-flow environment reallocation based on scheduling strategies. The nature of distributed computational environments itself demands the development of multicriteria and multifactor strategies [17, 18] of coordinated scheduling and resource allocation.

The dynamic configuration of the environment, large number of resource reallocation events, user’s and resource owner’s needs as well as virtual organization policy of resource assignment should be taken into account. The scheduling strategy is formed on a basis of formalized efficiency criteria, which sufficiently allow reflecting economical principles [14] of resource allocation by using relevant cost functions and solving the load balance problem for heterogeneous processor nodes. The strategy is built by using methods of dynamic programming [20] in a way that allows optimizing scheduling and resource allocation for a set of tasks, comprising the compound job. In contrast to previous works, we consider the scheduling strategy as a set of admissible supporting schedules (see Fig. 2, Tables 2 and 3). The choice of the specific variant depends on the load level of the resource dynamics and is formed as a resource query, which is sent to a local batch-job processing system.

One of the important features of our approach is resource state forecasting for timely updates of the strategies. It allows implementing mechanisms of adaptive job-flow reallocation between processor nodes and domains, and also means that there is no more fixed task assignment on a particular processor node. While one part of the job can be sent for execution, the other tasks, comprising the job, can migrate to the other processor nodes according to the updated co-allocation strategy. The similar schedule correction procedure is also supported in the GrADS project [13], where multistage job control procedure is implemented: making initial schedule, its correction during the job execution, metascheduling for a set of applications. Downside of this approach is the fact, that it is based on the creation of a single schedule, so the metascheduler stops working when no additional resources are available and job-queue is then set to waiting mode. The possibility of strategy updates allows user, being integrated into economical conditions of virtual organization, to affect job start time by changing resource usage costs. In fact it means that the job-flow dispatching strategy is modified according to new priorities and this provides competitive functioning and dynamic job-flow balance in virtual organization with inseparable resources.

### 4 SIMULATIONS STUDIES AND RESULTS

#### 4.1 Simulation System

We have implemented an original simulation environment (Fig. 7) of the metascheduling framework (see Fig. 6) to evaluate efficiency indices of different scheduling and co-allocation strategies. In contrast to well-known Grid simulation systems such as ChicSim [12] or OptorSim [23], our simulator MetaSim generates...
multicriteria strategies as a number of supporting schedules for metascheduler reactions to the events connected with resource assignment and advance reservations.

Strategies for more than 12000 jobs with a fixed completion time were studied. Every task of a job had randomized completion time estimations, computation volumes, data transfer times and volumes. These parameters for various tasks had difference which was equal to 2, ..., 3. Processor nodes were selected in accordance to their relative performance. For the first group of “fast” nodes the relative performance was equal to 0.66, ..., 1, for the second and the third groups 0.33, ..., 0.66 and 0.33 (“slow” nodes) respectively. A number of nodes was conformed to a job structure, i.e. a task parallelism degree, and was varied from 20 to 30.

4.2 Types of Strategies

We have studied the strategies of the following types:
- S1 – with fine-grain computations and active data replication policy;
- S2 – with fine-grain computations and a remote data access;
- S3 – with coarse-grain computations and static data storage;
- MS1 – with fine-grain computations, active data replication policy, and the best- and worst execution time estimations (a modification of the strategy S1).

The strategy MS1 is less complete than the strategy S1 or S2 in the sense of coverage of events in distributed environment (see Tables 2 and 3). However the important point is the generation of a strategy by efficient and economic computational procedures of the metascheduler. The type S1 has more computational expenses than MS1 especially for simulation studies of integrated job-flow and application-level scheduling.

Therefore, in some experiments with integrated scheduling we compared strategies MS1, S2, and S3.

4.3 Application-Level Scheduling Study

We have conducted the statistical research of the critical works method for application-level scheduling with above-mentioned types of strategies S1, S2, S3. The main goal of the research was to estimate a forecast possibility for making application-level schedules with the critical works method without taking into account independent job flows. For 12000 randomly generated jobs there were 38% admissible solutions for S1 strategy, 37% for S2, and 33% for S3 (Fig. 8). This result is obvious: application-level schedules implemented by the critical works method were constructed for available resources non-assigned to other independent jobs.

Along with it there is a conflict distribution for the processor nodes that have different performance (“fast” are 2-3 times faster, than “slow” ones): 32% for “fast” ones, 68% for “slow” ones in S1, 56% and 44% in S2, 74% and 26% for S3 (Fig. 9). This may be explained as follows. The higher is the task state of distribution in the environment with active data transfer policy, the lower is the probability of collision between tasks on a specific resource.

In order to implement the effective scheduling and resource allocation policy in the virtual organization we should coordinate application and job-flow levels of the scheduling.

Figure 7: Simulation environment of hierarchical scheduling framework based on strategies.
Figure 8: Percentage of admissible application-level schedules.

4.4 Job-Flow and Application-Level Scheduling Study

For each simulation experiment such factors as job completion “cost”, task execution time, scheduling forecast errors (start time estimation), strategy live-to-time (time interval of acceptable schedules in a dynamic environment), and average load level for strategies $S_1, S_2, S_3$, and $S_3$ were studied.

Figure 10 shows load level statistics of variable performance processor nodes which allows discovering the pattern of the specific resource usage when using strategies $S_1, S_2$, and $S_3$ with coordinated job-flow and application-levels scheduling.

The strategy $S_2$ performs the best in the term of load balancing for different groups of processor nodes, while the strategy $S_1$ tries to occupy “slow” nodes, and the strategy $S_3$ - the processors with the highest performance (see Fig. 10).

Figure 9: Percentage of collisions for “fast” processor nodes in application-level scheduling.

Figure 10: Processor node load level in strategies $S_1, S_2, S_3$.
selected processor nodes as well as modification MS1, when best- and worst-case execution time estimations were taken, is shown in Figures 11 and 12.

![Figure 11: Job completion cost and task execution time in strategies MS1, S2, and S3.](image1)

Lowest-cost strategies are the “slowest” ones like S3 (see Fig. 11); they are most persistent in the term of time-to-live as well (see Fig. 12).

![Figure 12: Time-to-live and start deviation time in strategies MS1, S2, and S3.](image2)

The strategies of the type S3 try to monopolize processor resources with the highest performance and to minimize data exchanges. Withal, less persistent are the “fastest”, most expensive and most accurate strategies like S2. Less accurate strategies like MS1 (see Fig. 12) provide longer task completion time, than more accurate ones like S2 (Fig. 11), which include more possible events, associated with processor node load level dynamics.

5 CONCLUSIONS AND FUTURE WORK

The related works in scheduling problems are devoted to either job scheduling problems or application-level scheduling. The gist of the approach described is that the resultant dispatching strategies are based on the integration of job-flows and application-level techniques. It allows increasing the quality of service for the jobs and distributed environment resource usage efficiency.

Our results are promising, but we have bear in mind that they are based on simplified computation scenarios, e.g., in our experiments we use FCFS management policy in local batch-job management systems. Afore-cited research results of strategy characteristics were obtained by simulation of global job-flow in a virtual organization. Inseparability condition for the resources requires additional advanced research and simulation approach of local job passing and local processor nodes load level forecasting methods development. Different job-queue management models and scheduling algorithms (FCFS modifications, LWF, backfilling, gang scheduling, etc.) can be used here. Along with it local administering rules can be implemented.

One of the most important aspects here is that advance reservations have impact on the quality of service. Some of the researches (particularly the one in Argonne National Laboratory) show, that preliminary reservation nearly always increases queue waiting time. Backfilling decreases this time. With the use of FCFS strategy waiting time is shorter than with the use of LWF. On the other hand, estimation error for starting time forecast is bigger with FCFS than with LWF. Backfilling that is implemented in Maui cluster scheduler includes advanced resource reservation mechanism and guarantees resource allocation. It leads to the difference increase between the desired reservation time and actual job starting time when the local request flow is growing. Some of the quality aspects and job-flow load balance problem are associated with dynamic priority changes, when virtual organization user changes execution cost for a specific resource.

All of these problems require further research.

ACKNOWLEDGEMENT. This work was supported by the Russian Foundation for Basic Research (grant no. 09-01-00095) and by the State Analytical Program “The higher school scientific potential development” (project no. 2.1.2/6718).

6 REFERENCES


