ISSN 1392 – 124X INFORMATION TECHNOLOGY AND CONTROL, 2010, Vol.39, No.4

A GENETIC ALGORITHM APPROACH FOR UTILITY MANAGEMENT SYSTEM WORKFLOW SCHEDULING

Nemanja Nedić, Srđan Vukmirović, Aleksandar Erdeljan, Lendak Imre, Darko Čapko

Faculty of Technical Sciences, University of Novi Sad
Novi Sad, Serbia
e-mail: nemanja.nedic@dmsgroup.rs, {srdjanvu, ftn_erdeljan, lendak, dcapko}@uns.ac.rs

Abstract. In this paper, we present a scheduler for distributing workflows in Utility Management System (UMS). The system executes a large number of workflows, which have very high resource requirements. The workflows have different computational requirements and thus the optimization of resource utilization must be performed in a way that is different from the standard approach of scheduling workflows. We developed a strategy for allocating workflows, which is based on a genetic algorithm. The proposed architecture executes a scheduling algorithm by using a feedback from the execution monitor. We also report on an experimental study, which shows that a significant improvement of overall execution time can be achieved by using the genetic algorithm. The algorithm is used for designing effective Grid schedulers that optimize makespan. The study further shows that the overall system (UMS) performance is significantly improved; this finding indicates that there can be reduction in hardware investment.

1. Introduction

Utility Management Systems (UMS) is a term used to refer to various systems, such as power / gas or water distribution systems. The UMS are becoming increasingly resource demanding because their scopes are becoming increasingly wider. The systems have some exceptional requirements such as: communication with end devices (sensors and actuators) and storage of huge time-series data volumes about variable values.

A workflow is loosely defined as an automation of a coordination process: coordinating people, data and tasks. A lot of research on business wokflows as well as their use have been done over the past few decades [7].

Foster and Kesselman described the Grid as an infrastructure that connects computers, databases, instruments, and people into a seamless web of advanced capabilities [1]. With the development of large-scale high-speed networks, the Grid has become an attractive computational platform for high-performance parallel and distributed applications, such as UMS.

Workflow scheduling is a decision process that assigns application components to available resources in order to optimize various performance metrics [20]. The Grid workload management and scheduling subsystems enable the efficient distribution of tasks and allow their transparent execution by hiding the complexity of the Grid infrastructure.

At present, in many modern Grid infrastructures, scheduling relies only on static properties and predetermined states of resources. We argue that resource utilization can be enhanced by adding the run time information. Also, it can be enhanced by predicting the system performance based on the current system information. For this reason, we propose an architecture for the Workflow Scheduler system that primarily uses the data of the current state of the Grid.

We achieve the enhancement of resource utilization performance in the Workflow Scheduler by carrying out workflow manipulation. The Workflow Scheduler can adopt various optimization criteria which it uses in order to choose the optimal workflow. Some possible criteria include the ones that rely on static information knowledge (e.g., type of workflow and pre-defined order of execution, etc.) and the ones that rely on dynamic information (e.g., database performance, processor statuses, communication with end device, etc.).

Criteria can pursue different goals: the minimization of a single task's execution time, the minimization of workflow execution time, the fairness of load distribution, maximum time of execution per workflow type, etc. Optimization rules are based on quantifiable metrics, such as: workflow reliability and distribution fairness, workflow average execution time, etc [3].

Artificial intelligence represents a modern concept of solving problems in engineering practice [19]. We chose a set of UMS workflows to analyze the feasibility and usability of applying genetic algorithm to this field. We used the accommodation of the genetic algorithm to solve the workflow scheduling. The experimetal study that we ran confirmed the usefulness of the genetic algorithm based scheduler when the makespan is optimized. The implementation of the genetic algorithm is extensively tested and the comparison with the previously provided solutions of similar problems shows that this scheduling system allocates workflows efficiently and more effectively; consequently, the performance improves.

2. Related Work

Grid computing is a new approach in scientific applications. Recent advances in grid infrastructure and middleware development have enabled various types of applications in science and engineering to be deployed on the Grid. The applications include those for climate modeling, computational chemistry, bioinformatics and computational genomics, remote control of instruments, and distributed databases [8].

The Grid infrastructure is used both to share expensive and centralized resources among many scientists, as well as to integrate experimental data sources with the simulation codes necessary to analyze them [9].

The Grid connects computers, databases and instruments in a seamless web, supporting rich computation application concepts such as distributed supercomputing, smart instruments, data-mining and complicated algorithm calculations [18]. However, its use has been limited to specialists, primarily because of the lack of usability [10].

Supervisory Control and Data Acquisition (SCADA) systems are becoming more and more resource demanding because their scope has become wider. This trend is especially visible in distribution systems for utilities – UMS, the systems that are an extension of the SCADA systems. SCADA systems went a long way from simple visualization of processes observed. Distributed SCADA systems are thoroughly described in a scientific paper [4]. The process industry data found their way to the Internet [5] and even cell phones [6].

An upcoming need for the Grid approach could be envisioned by observing the volumes of data that need to be stored and processed. This is especially apparent in the UMS; for instance, the number of process variables exceeds tens of thousands in Distribution Management Systems (DMS) [11].

Artificial intelligence slowly finds its purpose in distribution systems for utilities. It will surely be the key feature for the resources manipulation in the Grid enviroment.

The genetic algorithm, as an evolutionary technique for large space search, proved to be a powerful tool for solving various problems [21,22]. Faced with a variety of situations, an intelligent Grid environment requires complex algorithms which will help manage the execution of different kinds of workflows. Several works [15] [16] [17] address the problems of Grid scheduling that adopt the genetic algorithm method for workflow manipulation in order to improve performance. The hybridization of the genetic algorithm with other heuristics (TS, SA) for dynamic workflow scheduling is presented in [14].

3. UMS Workflows

We found four types of grid nodes by analyzing the architecture and requirements for large scale distributed UMS systems. These are:

Processing node (PRN): its task is business calculation and data pre-processing, mainly for providing reports and an offline analysis of the system.

Objects database node (ODN): it is used for storing the static data gained from the distributed UMS system. It usually hosts a relational database for better search performances.

Time-series node (TSN): this node hosts data about fast changing values of process variables.

Communication node (CON): this node is responsible for communication with end devices.

The workflows used for testing, which are reported in this paper, are chosen from the real UMS use cases. The following conditions are implied:

1. All workflows are independent of each other.
2. All workflows have the same priority.
3. Every node processes only one workflow at a time.
4. Every workflow is processed at one node at a time.
5. A workflow of the same type has the same execution time at each specified node.

We implemented five workflows that use different types of Grid nodes:

1. **Direct Command**: it sends commands to actuators. When executing, it will send commands to actuators through **CON,** and write command results to time series database in **TSN.**
2. **Command with pre-processing:** this type of command needs pre-processing of data prior to sending commands. The commands could be used when business logic has to be applied before actuator could be used. In this workflow, **PRN** first prepares the data, and after that, **CON** sends commands to actuators.
3. **Read Variable values:** in this scenario Variables are read from cache (previously read from devices). Workflows execution of this type starts in **ODN** in order to filter end devices that should be read. After that, the execution is transferred to **TSN** to read values of the selected Variables.

4. **Read Variable values From Device:** workflows of this type, contrary to the type 3 workflow, have to read values from sensors on demand. When executing, this workflow is transferred to **ODN** in order to filter end devices that should be read. After that, **CON** sends an on demand reading command to sensors.
5. **Reporting Inquiry:** this workflow covers various types of data processing. During the execution, this workflow is transferred to **ODN** in order to retrieve data needed for calculation. After that, it is transferred to **PRN,** which is responsible for calculation.

Figure 1 presents the execution plan of workflow migration for a set of previously defined UMS workflows. Optimization goal in this paper is to rearrange incoming workflows in order to get maximum usage of all nodes.

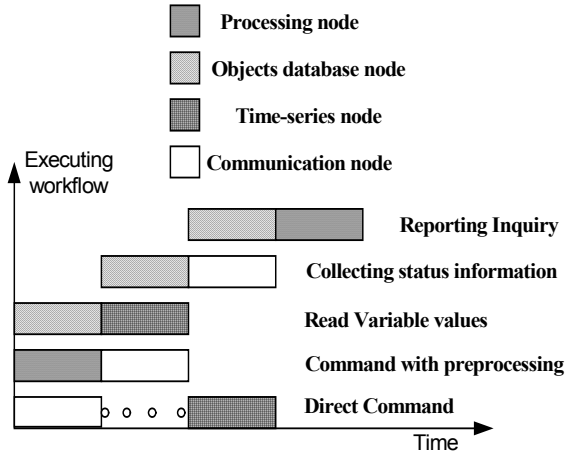

Figure 1. Workflow execution migration between nodes

3. Proposed Architecture

The proposed architecture takes into account the dynamic nature of a real-world UMS and uses the Grid environment approach for detecting and responding to the environmental change in the UMS. The developed framework is presented in Figure 2. It provides required support for the feedback from the scheduling process.

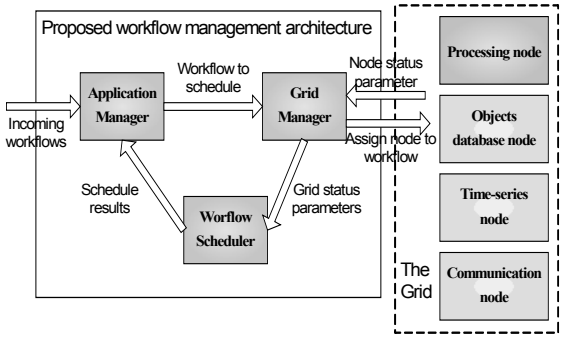

Figure 2. Proposed architecture of the system

The framework consists of three components: the Grid Manager, the Workflow Scheduler, and the Application Manager. The function of the **Grid Manager** is to monitor the status of the control variables and to send workflows to the nodes for execution. The **Workflow Scheduler** is responsible for resource selection and for mapping the workflows to the resources. This component is of the most interest for our research, since decision-making takes place in it. The **Application Manager** receives workflows in run time, queues them and works with Workflow Scheduler to determine the right time to send a workflow to execution in the Grid.

4. Model representation in Genethic Algorithm

Genetic algorithms (GA) are stochastic, evolution-based, search and optimization algorithms. A potential solution of the optimization problem is encoded in an artificial creation, which is called chromosome (individual). The population of chromosomes evolves into a better solution in each iteration of the algorithm. The fittest individuals survive and are able to exchange their genetic material. Thus in every new generation, a set of chromosomes is created by exploiting the information from the past generation.

A. Chromosome encoding

A chromosome consists of a series of genes. The solution proposed in this paper presents an optimized workflow sequence, which is sent to the Grid for the execution. Each gene thus represents a workflow type. All of the workflow types are described in section 3 (1 – Direct Command, 2 – Command with pre-processing, 3 – Read Variable values, 4 – Read Variable values From Device, 5 – Reporting Inquiry). A gene is an integer value in the interval from 1 to 5.

B. The fitness fuction

The essence of the GA is the following: it searches for and eventually finds an optimal solution to a problem by creating new generations and evaluating the individuals. It is necessary to define the principle of the individual assessment. The fitness function [2] provides the mechanism to evaluate each individual in the problem domain.

The problem that we offer a solution for in this paper is to maximize the throughput of the Grid by balancing the load among the nodes in an intelligent way. The fundamental optimization criterion in solving this kind of the problem is the minimization of the *makespan*, i.e. the time when the latest workflow is finished. This criteria, for an individual with *n* workflows, is represented in the following way:

$$makespan = \max\{Wft_i\}, \quad (1)$$

$$makespanMin = \min_{s \in Sequneces}\{makespan_s\}. \quad (2)$$

Wft_i represents time when the i^{th} workflow in the workflow sequence is finished, and the *Sequences* represent all possible sequences (schedules) of *n* workflows.

A conclusion is then, that the time when the last workflow ends corresponds to the time of the completion of the last node. So, the time when the last node finishes its work is the *makespan*.

If we define $nodeStartTime_k$ as a time point when a node $k, k \in \{1..numberOfNodes\}$ is ready to start executing assigned work, and $nodeWorkTime_k$ as required time to get the work done, then the predicted completion time of a node is:

$$nodeEndTime_k = nodeStartTime_k + \\ + nodeWorkTime_k. \quad (3)$$

The workflows described in this paper consist of two tasks, and each of these tasks executes at an appropriate node. In order to formulate the predicted time for a node to complete its work (*nodeWorkTime*), we need to estimate the execution time of the tasks. A computational load of tasks and computing capacity of nodes are required. The estimation is easily determined in practice; it is easy to know a computing capacity of a resource (node) by knowing its characteristics and, the workload of each task can be evaluated from the history data. The task execution time is calculated by dividing the workload of the task (in millions of instructions – *mips*) with the computing capacity of the node (in *mips*). Calculation of the workload of tasks is presented in the Cornell Theory Center [12].

A workflow can be defined as a set of two tasks (primary and secondary). The execution time for each task is:

$Tp_{i,k}$ – execution time for the primary task of a workflow *i* which is executed on the node *k*,

$Ts_{i,k}$ – execution time for the secondary task of a workflow *i* which is executed on the node *k*.

Predicted time for a node to complete its work is defined as:

$$nodeEndTime_k = nodeStartTime_k + \\ + \sum_{i,j=k} Tp_{i,j} + \sum_{i,j=k} Ts_{i,j}. \quad (4)$$

The *makespan* is determined as:

$$makspan = \max\{nodeEndTime_k\}, \quad (5)$$

$$k \in \{1..numberOfNodes\}.$$

In the proposed implementation of the genetic algorithm, the fittest individuals have the highest numerical values for the fitness function, defined as:

$$ff = \frac{1}{makespan} \quad (6)$$

C. Algorithm

The approach used in this paper generates a set of initial individuals (sequences, schedules of workflows), estimates the fitness gain, selects the most appropriate individuals and combines them using the operators (crossover and mutation) in order to form new solutions.

A selection mechanism guarantees the survival of the fittest individuals. A part of the existing population is selected to breed; it is highly likely that fitter individuals survive and receive a higher number of descendents. A roulette wheel mechanism [2] is used to construct a proportional selection. This mechanism selects a small part of the less fit individuals and thus keeps the diversity of the population; this prevents a premature convergence towards a poor solution.

A crossover operator is the most important ingredient of the genetic algorithm. It produces new individuals by interchanging parents' genetic material. Its aim is to obtain better quality descendents and to explore new parts of the solutions space that have not been considered so far.

Many types of the crossover operators are demonstrated in literature: one-point, k-point, uniform crossover [2], etc. We decided to use the k-point crossover. This operator provides a thorough study of the solutions space; however, it increases the possibility of destroying the parents' structure. In order to preserve the genetic material of the parents, we decided that the two fittest individuals among parents and descendants join the next generation after the crossover is performed.

Since the crossover operator is dominant in the genetic algorithm, the crossover rate (a probability that the two chosen parent individuals will be crossed) needs to be set to a high value. An experimental study showed that best results are obtained with the crossover rate of 0.85.

Mutation, an operator that changes the individual's gene to another allowed value, is randomly applied with a low probability. It is used to ensure genetic diversity of the population and to recover good genetic material that may be lost during the crossover and the selection.

The mutation operator is used in the following manner: after the crossover is performed and the two individuals are chosen to join the next generation, the mutation operator is applied with the probability of 0.05 (mutation rate). The operator is implemented to a randomly selected individual's gene. The value of the gene is replaced with a random value taken from the set of the possible gene values.

The strategy of creating the next generation that allows the entry of the fittest individuals from the previous generation is known as elitism. This variant of the genetic algorithm can be very successful. At the same time, the elitism should be carefully used since there is a possibility that the algorithm gets stuck in a local extreme. We reduced the elitism in our research to 5%.

The pseudo code for the genetic algorithm implementation is shown below (Listing 1):

```
begin
 initialization:
 Generate initial population of s individuals
 while(count of generation is less then specified)
{
 Evaluate fitness function for individuals
 Create empty next generation
 elitism: specified percents of fittest individuals
 Choose m fittest individuals → ind1, ind2,..., indm
 Add to next generation (ind1, ind2,..., indm)
 for ( 1 .. s - m)
 {
 selection: roulette wheel mechanism
Choose two parents from current generation →
 → parent1, parent2
 crossover: crossover rate
 Crossover(parent1, parent2) → child1, child2
 Choose the fittest two from
(parent1, parent2 ,child1, child2) → res1, res2
 mutation: mutation rate
 Mutation(res1)
 Mutation(res2)
 Add to next generation (res1, res2)
 }
 current generation → next generation
}
 return Best individuals from current generation
 end
```

Listing 1. Genetic algorithm implementation pseudo code

6. GA accommodation for Workflow Schedular

Since the workflows arrive continuously and the set of those to be executed on the Grid is not known in advance, all pieces of information regarding the Grid tasks are not entirely known before the execution time. Therefore, scheduling decisions must be made on the fly, and a dynamic scheduling algorithm is necessary.

One of the main disadvantages of the genetic algorithms is that they need a significantly longer period of time to generate a solution; whereas some other concepts of artificial intelligence (e.g. neural networks) do it in a much shorter period. This time period is a consequence of the iterative process of creating individual generations. It is impractical and unviable for the dynamic scheduler to calculate the fittest individual for any decision on a future workflow. We propose the following:

A. Dinamic schedular algorithm

At startup, the dynamic workflow scheduler uses the genetic algorithm (described in Section 5) to generate an optimal sequence of *n* workflows (an individual with *n* genes). After the first workflow is sent to the Grid, the sequence contains *n–1* workflows. The dynamic workflow scheduler uses the following steps to add a new workflow at the end of the sequence so that a sequence remains optimal (Figure 3):

1. reads the remaining types of workflows from the workflows queue,
2. creates an *n*-dimensional sequence by adding a workflow at the end of the sequence for each type of the remaining workflows,
3. calculates a fitness function for every resulting sequence,
4. keeps the sequence with the highest fitness function as a new optimal sequence.

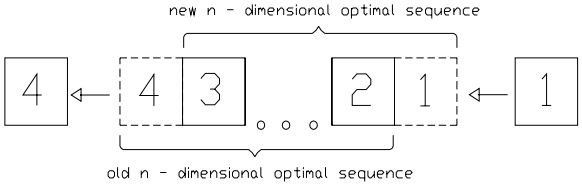

Figure 3. *n* – sequence optimization

B. Calculation of fitnes fuction during dinamic schedular algorithm

The dynamic workflow scheduler obtains information related to the current Grid load after the first workflow from the optimal sequence is sent to execution. The time periods predicted for the nodes to complete their work are collected and they are used as *nodeStartTime* for fitness function calculation of the processed sequence*;* it signals when the nodes are ready to start executing assigned work (Section 5). The workflows in the processed sequence are defined and the execution times of their primary and secondary tasks are calculated. All information regarding the fitness function for the processed sequence is available and the fitness function can be calculated.

7. Results and discussion

We developed a distributed testing environment based on the proposed architecture. Each type of the nodes specific for the UMS systems is attributed a computer node (described in Section 2). The scheduling application controls the execution of the workflows queue. Figure 4 shows the deployment strategy. Each of the four types of nodes inside the Grid is hosted in a separate computer. All the software components that we argue for in the paper are hosted in a separate computer.

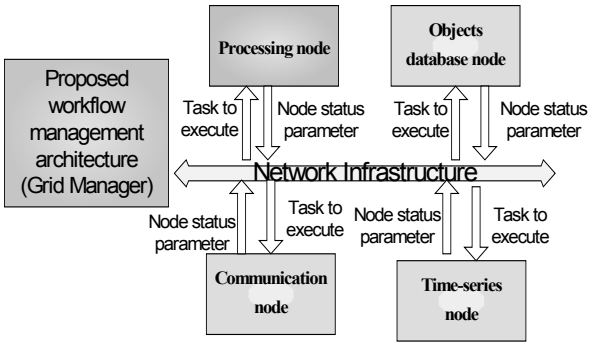

Figure 4. Deployment diagram for test environment

In this experiment we used a benchmark of queues of the five UMS workflows types defined in Section 3. We did this in order to study the performance of the dynamic GA workflow scheduler presented. We compared the quality of the results produced by the scheduler with the results reported in the literature [13]. The research presented in the literature [13] describes the following scheduling logic: *No optimization* – workflows are scheduled in the order in which they arrive in the input queue; *Simple ANN optimization* – the output of the simple ANN presents the workflow that is sent to the Grid; *Hierarchical ANN optimization* – the output of the hierarchical ANN network selects the workflow that is scheduled in the next step.

We tested the dynamic workflow scheduler based on GA with parameters given in Table 1.

Table 1. Values of parameters used in GA scheduler

population size	100
number of evolution steps	3000
chromosome size	6
selection	roulette wheel mechanism
crossover operator	k-point crossover
crossover rate	0.85
mutation rate	0.05
elitism	5 %

Table 2 summarizes the results of scheduling comparison. We used between ten and a thousand workflows in the tests.

Table 2. Speed of workflows execution

Number of workflows	Time of execution – no optimization [s]	Time of execution – simple ANN optimization [s]	Time of execution – Hierarchical ANN optimization [s]	Time of execution – GA accommodation optimization [s]
10	17	14	13	16
50	99	77	72	67
100	199	163	156	128
250	501	412	391	324
500	1007	827	809	618
1000	2018	1671	1625	1297

The experimental study shows that the GA based scheduler outperforms the existing solutions based on the neural networks (Figure 5). The performance (specifically, the speed) of the system improves with the increasing number of the scheduled workflows. The scheduling brings more benefits if there are more workflows to rearrange.

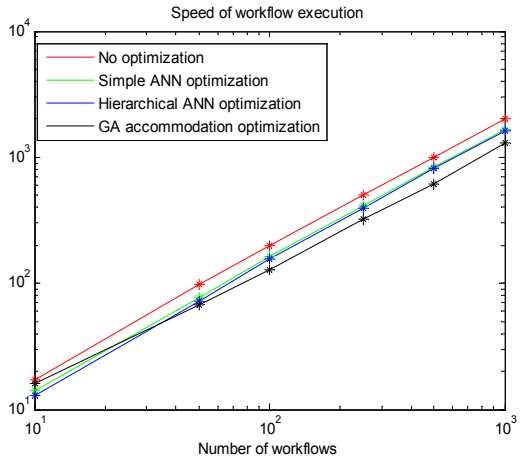

Figure 5. Speed of workflow execution

By introducing the dynamic GA scheduler in a workflow scheduling process for large scale UMS system, we provide the substantial improvement of the computational resources exploitation. The result is better performance of the entire system.

8. Conclusion

The specific features of the large scale UMS make the proposed architecture different from the standard Grid scheduling systems. The architecture based on feedback provides optimal scheduling (execution time optimization) of the workflows.

We demonstrated the usefulness of the genetic algorithm in forming the efficient workflow Grid scheduler. The aim of the experimental study was to reveal the effectiveness of the GA based scheduling when the *makespan* is optimized.

The GA scheduler works fast because of the GA Accommodation algorithm; hence the scheduler dynamically chooses the optimal scheduling strategy for the workflows that arrive at the Grid.

We also presented an experimental study. The performance analysis shows that this approach significantly boosts the performance of the whole system and reduces the total execution time. Since the same results can be achieved by using less Grid nodes, the hardware investments can be substantially decreased.

References

[1] **I. Foster, C. Kesselman, S. Tuecke.** The anatomy of the grid: enabling scalable virtual organization. *International Journal of Supercomputer Applications*, 2001, 15(3), 200–222.

[2] **V. Kecman.** Learning and Soft Computing. 2001.

[3] **Y. Zhang, C. Koelbel and K. Cooper.** Hybrid Re-Scheduling Mechanisms for Workflow Applications on Multi-cluster Grid. 9*th IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2009, 116–123.

[4] **R. K. Agrawal, B. Merh, P. Fatnani, R. Yadav, S. Gangopadhyay.** Scada functionality for control operations of Indus-2. 10*th ICALEPCS Int. Conf. on Accelerator & Large Expt. Physics Control Systems. Geneva*, 10–14, *Oct*. 2005, PO1.098-8.

[5] **B. Qiu H. B. Gooi.** Web-Based SCADA Display Systems (WSDS) for Access via Internet. *IEEE Transactions on Power Systems*, 2000, *Vol*.15, 681–686.

[6] **E. Ozdemir, M. Karacor**. Mobile phone based SCADA for industrial automation. *ISA Transactions*, 2006, *Vol*.45, 67–75.

[7] **A. Rygg, S. Mann, P. Roe, O. Wong.** Bio-Workflows with BizTalk: Using a Commercial Workflow Engine for eScience. *Proceedings of the First International Conference on e-Science and Grid (e-Science'05)*, 2005, 116–123.

[8] **D. Abramson, A. Lynch, H. Takemiya, Y. Tanimura et. al.** Deploying scientific applications to the PRAGMA grid testbed: Strategies and lessons. *In Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, Washington, *DC, USA*, 2006, *IEEE Computer Society*, 241–248.

[9] **G. Allen, D. Angulo, T. Goodale, T. Kielmann, A. Merzky, J. Nabrzyski, J. Pukacki, M. Russell, T. Radke, E. Seidel, J. Shalf, I. Taylor.** GridLab: Enabling applications on the Grid. *In GRID*'02: *Proceedings of the Third International Workshop on Grid Computing, London, UK*, 2002, *Springer-Verlag*, 39–45.

[10] **F. Berman, H. Casanova, A. Chien, K. Cooper et. al.** New grid scheduling and rescheduling methods in the grADS project. *Int. J. Parallel Program*, 2005, 33(2–3), 209–229.

[11] **A. Erdeljan, N. Trninić, D. Čapko.** An OPC data access server designed for large number of items. 6*th International Symposium Interdisciplinary Regional Research (Hungary, Romania, Yugoslavia) ISIRR* 2002, *Novi Sad*, 2002.

[12] **S. Hotovy.** Workload evolution on the Cornell theory center IBM SP2. *In Job Scheduling Strategies for Parallel Processing Workshop*, 1996, 27–40.

[13] **S. Vukmirovic, A. Erdeljan, I. Lendak, N. Nedic.** Hierarchical neural model for workflow scheduling in Utility Management Systems, 2010.

[14] **A. Abraham, R. Buyya, and B. Nath.** Nature's heuristics for scheduling jobs on computational grids. *In The* 8*th IEEE International Conference on Advanced Computing and Communications* (*ADCOM* 2000), *India*, 2000, 45–52.

[15] **V. Di Martino, M. Mililotti.** Sub optimal scheduling in a grid using genetic algorithms. *ParallelComputing*, 2004, 30, 553–565.

[16] **M. Aggarwal, R.D. Kent, A. Ngom.** Genetic Algorithm Based Scheduler for Computational Grids. *Proceeding of the* 19*th Annual International Symposium on High Performance Computing Systems and Applications*, 2005, 209–215.

[17] **U. Fissgus.** Scheduling Using Genetic Algorithms. 20*th IEEE International Conference on Distributed Computing Systems* (*ICDCS*'00), 2000, 662–699.

[18] **A. Kačeniauskas, R. Pacevič, A. Bugajev, T. Katkevičius.** Effective visualization by using Paraview software on Balticgrid. *Information Technology and Control*, 2010, *Vol*.39, *No*.2, 108–115.

[19] **G. Narvydas, R. Simutis, V. Raudonis**. Autonomous mobile robot control using If-Then rules and genetic algorithm. *Information Technology and Control*, 2008, *Vol*.37, *No*.3, 193–197.

[20] **A. Kuczapski, M. V. Micea, L. A. Maniu, V. I. Cretu.** Efficient generation of near optimal initial populations to enhance genetic algorithms for job-shop scheduling. *Information Technology and Control*, 2010, *Vol*.39, *No*.1, 32–37.

[21] **A. Misevičius, D. Rubliauskas.** Enhanced improvement of individuals in genetic algorithm. *Information Technology and Control*, 2008, *Vol*.37, *No*.3, 179–186.

[22] **A. Misevičius, D. Rubliauskas, V. Barkauskas.** Some further experiments with the genetic algorithm for the quadratic assignment problem. *Information Technology and Control*, 2009, *Vol*.38, *No*.4, 325–332.

Received June 2010.

DOI: 10.5755/j01.itc.39.4.12387