ISSN 1392 – 1215 ELECTRONICS AND ELECTRICAL ENGINEERING 2012. No. 4(120)
ELEKTRONIKA IR ELEKTROTECHNIKA

T 120 SYSTEM ENGINEERING, COMPUTER TECHNOLOGY
SISTEMŲ INŽINERIJA, KOMPIUTERINĖS TECHNOLOGIJOS

Dynamic Repartitioning of Large Data Model in Distribution Management Systems

D. Capko, A. Erdeljan, G. Svenda, M. Popovic
Faculty of Technical Sciences,
Trg Dositeja Obradovica 6, 21000 Novi Sad, Serbia, phone: +381 63 1028061, e-mail: dcapko@uns.ac.rs

crossref http://dx.doi.org/10.5755/j01.eee.120.4.1461

Introduction

Modern large-scale critical infrastructure systems (such as electric power, water and gas management systems) are faced with a continuous increasing amount of data that should be processed and calculated. One way to optimize functionality of the systems is parallelization of their calculations [1]. In this way, the data model is shared among the processors so that the parts in terms of the calculations are independent. Bearing in mind that the divided data can be dynamically changed, causing a change of independent groups of data, it is necessary to reallocate the already partitioned data. In this paper, algorithms for dynamic data repartitioning that are adjusted for NUMA multiprocessor architecture are discussed.

NUMA multiprocessor computer is organized in nodes in which each node has a set of processors and part of the main memory. The processor topology determines the memory access time to data, and therefore recognizes the different levels of access to data.

This paper analyzes data models in DMS system. These systems contain and process data on the distribution power network to perform the supervision, management and planning of electric power network. DMS systems are composed of the following components: 1) SCADA system that takes the value of remote terminal units (RTUs) and sets dynamic values; 2) DMS analytical functions that are performed in order to optimize the operation, planning and operation of networks in alarm situations; 3) a database that stores data and history of data changes in the system. Following the dynamics of change of certain parameters in the system (such as the change in status of switches) DMS system needs to promptly determine the state of the system and possibly initiate some control actions.

The data model of such systems represents radial weakly meshed networks that are suitable to be divided into smaller sub-models, which will be used by parallel tasks as independent data model partitions. The aim of partitioning is to optimally divide data in order to minimize data relations across these subsets. Also, by making partitions of the similar size the computation load of every processor in the multiprocessor system might be balanced. In order to achieve this balance, graph based partitioning methods are used [2]. Therefore, a graph is created out of the data model and the optimization problem is set as a problem of graph partitioning.

The optimal partitioning can be done in two complementary ways: 1) initially – before starting the system and 2) dynamically – while the system works (on-line) [3]. The dynamic repartitioning is applied while the system is working (on line) – triggered by changes in inputs (that affects the calculations) or periodically if the processor load is imbalanced [6]. When such imbalance is detected, the dynamic repartitioning is started and data are migrated among processors' local memories.

Cybenco [4] used the diffusion method for dynamic repartitioning. Following his research other diffusion algorithms for dynamic load balancing were developed. In [5] the diffusion algorithm for dynamic load balancing is described. In that algorithm repartitions are defined by calculating the Lagrange multiplier and using Laplacian matrix. The most studied techniques of dynamic load balancing are scratch-remap and diffusion algorithms [6, 7], which minimize data migration by extending load balancing and number of connections between partitions (external edges) with additional optimization criteria. Diffusion algorithms show better results than scratch-remap [6, 7] because they reduce data migration and the external edges. More recent diffusion algorithms described in [8] make further reductions in data migration and the external edges. However, they are slow and not applicable to on-line systems. Finally, the CP algorithm was studied because it requires the least data migrations.

In the case of NUMA multiprocessor systems, different levels of memory and speed of data access are discussed depending on the system architecture. In [9] multilevel algorithms for load balancing processors are studied. Different memory access levels are considered and balancing is carried out first among processors with the same level of memory access and then processors on the following memory access levels. According to this, specialized multilevel algorithms for dynamic graph partitioning in NUMA multiprocessor systems are developed.

In this paper, simple Diffusion Repartitioning (DR) and cut-paste (CP) algorithms for dynamic partitioning are applied. Additionally, modified versions of these algorithms (MDR and MCP) are developed to support dynamic repartitioning running in NUMA multiprocessor systems. MDR and MCP algorithms are giving better experimental results than DR and CP algorithms.

The content of the paper is the following: In Section 2, the problem and used terminology is described. The details of the discussed data model and definition of the optimization problem are presented as well; Section 3 describes proposed algorithms for dynamic repartitioning; Section 4 describes experimental setup, presents and discusses the results. Section 5 is a conclusion.

Problem definition

It is expected that control and supervision of the DMS system results in a quick respond to changes of certain parameters and calculations of necessary analytical DMS functions. The most significant DMS functions are: Topology analysis; Load Flow, State Estimation, Volt/Var Control, etc. This paper will focus on the optimization of functions with the calculations carried out on parts of the network (Load Flow and State Estimation are the most often used functions). Bearing in mind simplicity it is assumed that only one function ξ is running in the system and it is needed to optimally partition the data model in order to finish data processing as quickly as possible.

In order to analyze the problem more comprehensively, the data model and optimization criterion for the data partitioning will be described.

Data Model. A connectivity model that contains basic data types important for the calculations is shown in Figure 1. The model is based on the Common Information Model (CIM) [10, 11] for the purpose of efficient calculation (and memory relaxation). CIM is the most important standard for power energy systems and it is published by the IEC (International Electrotechnical Commission) as a part of their international standard IEC 61970-301 [10]. The CIM data model is an abstract object model that represents all major entities (and relations among them) in an electric utility enterprise. The data model is based on the related objects with preserved relationships between objects. Each object models one type of the electric element, and it contains all instances of the object type together with their attributes.

The connectivity model is composed of transformer substations (represented by *PowerSystemResource*) connected with power lines (*ACLineSegment*), and they are used to supply groups of consumers (*EnergyConsumer*). Substations contain various equipment (*ConductionEquipment* and *PowerTransformer*) and nodes (*ConnectivityNode*) that connect such equipments. *ConductionEquipment* objects are modelled with single- or double-ended conductors, and these ends are always connected with *ConnectivityNode*(s). For example, typical types of single-ended conductors are *EnergyConsumer*, *EnergySource,* and *BusbarSection*, while types of *Switch* (*Breaker*, *Fuse*), *ACLineSegment*, and others, are double-ended conductors. In essence, the connectivity model is a branch-node model suitable for a graph presentation, where edges and vertices are instances of *ConductionEquipment* and *ConnectivityNode*, respectively.

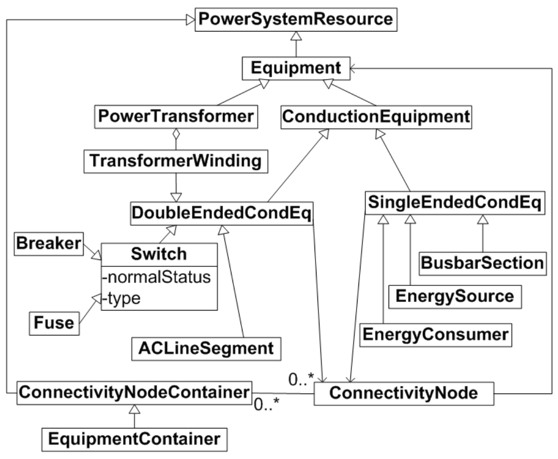

Fig. 1. Connectivity model

If two elements α_i and α_j are in relation $N(\alpha_i, \alpha_j)$ it is assumed that the processing function ξ will use them together (only relations meaningful to ξ are considered). This also means that the elements are connected and they are called *neighbours*.

Relationships between the elements can be temporary and specific to their state (which can be changed externally). The relation between neighbours that depends on an input value u_{ij} (for example, switch state) could be temporarily inactive and it is called *potential connection* $Pot(\alpha_i, \alpha_j, u_{ij})$. In other words, relations between elements depend on input values, i.e. when a potential connection is activated two elements become neighbours [3]

$$(\forall \alpha_i, \alpha_j \in Q)\; Pot(\alpha_i, \alpha_j, u_{ij} = active) \Rightarrow N(\alpha_i, \alpha_j) \quad . \quad (1)$$

Therefore, the potential connection is a relation that is characterized by the state (active or inactive). If the state is active then the elements are connected and they must be used for calculation together. In opposite, if the state is inactive, elements are not connected.

The set of mutually connected elements is called *calculation region* R_k (or just region)

$$(\forall \alpha_i \in R_k)\; N(\alpha_i, \alpha_j) \Leftrightarrow \alpha_j \in R_k, \quad k \in \{1,2,...,n\}, \qquad (2)$$

which is the smallest data unit that can be processed by ξ.

An overall connectivity data model of a radial or weakly meshed power distribution network presented as a graph is suitable for parallel calculations. The most commonly used power function is the Load Flow (LF) [12], which uses all galvanic connected electrical elements that are powered from the same source (*EnergySource*). Therefore, LF is the discussed calculation function, and elements that are used in the calculations are the electric elements. A set of all galvanically connected elements, which are simultaneously used in the LF calculation, is called a *root*, and builds a *region* as a vertex of the coarse graph. Region weight is equal to the number of elements in the given *root*. Therefore, open switches, making these edges potential connections (which are in the inactive state), determine the boundary between regions (graph vertices). The two regions are united by closing the boundary switch (active state) and the calculation region is expanded to contain these two regions. The number of open switches between two regions determines the weight of the edge between the graph vertices, i.e. *regions*.

Data Model Partitioning. In a multiprocessor environment, function ξ can be applied to individual regions in parallel. If the number of regions is bigger than the number of processors, regions are grouped into p partitions (the result of partitioning is the set of partitions $\Pi=\{\pi_1, \pi_2,\ldots, \pi_p\}$) and distributed to p processors. This implies that a single processor sequentially executes the function ξ against the whole partition.

Regions could change over time because connections among elements depend on input data (1). When a potential connection between two elements from different regions is activated, these two regions have to be merged. It is also expected that a deactivated potential connection could cause splitting of the region. Dynamic changes of inputs (1) cause activation/deactivation of potential connections which makes a number of potential connections between regions variable. The quantitative indicator of probability to connect two regions can be the number of potential connections between their elements, and it is used as an optimization criterion for grouping regions into partitions. Therefore, the potential connections can be treated as a possibility for energizing two elements from the same source.

Data about the connectivity model are used to make the initial graph. As mentioned before, the structure of this data model is organized as a set of radial networks that can be weakly meshed [1,12], and the coarsening of the graph is completely established on the physical structure of the system. Consequently, the size of the coarsened graph is much smaller than the size of the initial graph and the number of potential connections is close to the number of regions, which is extremely important for efficient graph partitioning.

In this research, the coarsening phase is applied first in order to group mutually connected elements into regions (2). The resulting domain D is described as a weighted undirected graph, $G=(V,E)$ made of vertices ($V=\{v_1,v_2,\ldots,v_n\}$) and edges ($E=\{e_1,e_2,\ldots,e_m\}$). The weight of the edge e_i is marked as $w(e_i)$ and the weight of vertex v_j is marked as $w(v_j)$. A vertex v_i represents region R_i with weight as the number of all its elements ($w(v_j) = w_{R_i} = |R_i|$). It is assumed that the vertex weight is directly related to the region's calculation complexity.

Graph edges represent potential connections between elements from different regions. Two regions, R_p and R_q, could have many potential connections and they are presented by using one edge $e(R_p,R_q)$ the weight of which is equal to the total number of such potential connection. This edge $e(R_p,R_q)$ represents a potential connection between regions $Pot_{Rp,Rq}$. If two regions have a potential connection they are called *neighbours*.

In order to define optimization criterion for a partition π_k, we need to define the partition weight W_{π_k}

$$W_{\pi_k} = \sum_{R_j \in \pi_k} w_{R_j}\ , \tag{3}$$

as the sum of all contained regions' weights. We also need to define a function ϕ_k as

$$\phi_k = \sum_{Rp,Rq \in \pi_k} Pot_{Rp,Rq}\ , \tag{4}$$

where regions R_p and R_q are in π_k. This function is an indicator of "good connectivity" between the regions in the partition.

First it is necessary to group regions into a defined number of partitions (p), so that the weights of these partitions are approximately the same. They can never be greater than the *maximal partition weight M* defined as: $M=(1+\varepsilon)\cdot\frac{1}{p}\sum_{k=1}^{p} W_{\pi_k}$, where W_{π_k} is weight of partition π_k, p is the number of partitions, and ε is the tolerance ($\varepsilon \in [0, (p-1)/p]$).

The optimization criterion should obtain the maximum connection inside a partition

$$F=\max\left(\sum_{k=1}^{p} \phi_k\right), \tag{5}$$

where function ϕ_k is given in (4), and all partition weights are constrained by: $W_{\pi_k} \le M, \forall k \in \{1,2,\ldots,p\}$.

It should be noted that the maximal number of connections inside each of the partitions means a minimal number of connections among partitions.

Two partitions π_x and π_y are *neighbours* if their regions are neighbours. Set of neighbouring partitions by the partition π_x is marked as $N(\pi_x)$.

Dynamic (on-line) Optimization. If some partitions have weights greater than the maximal partition weight ($W_{\pi_k} > M$), the graph is *imbalanced*. In an imbalanced graph, a partition is *overbalanced* if its weight is greater than maximal partition weight (M). The graph is *balanced* when no partition is overbalanced [6, 7].

Dynamic repartitioning is needed when an imbalance between the processors' load is detected, which was caused by changes of region weights. This could happen when:

1) changes of inputs activate potential connections between the regions from the different partitions, or
2) changes of some other inputs make processing in certain regions more frequent (this case is not considered in the paper).

In both cases, the calculation of region weights is required and the dynamic algorithm for region repartitioning is started. The on-line repartitioning task attempts to get balanced and optimally connected partitions by applying minimal migrations of the regions between the partitions.

Repartitioning of Regions Caused by Outer Influences. This is done when an input signal activates the potential connection between the regions. These regions are joined into the new region whose weight equals to the total weight of two related regions and all their edges are preserved. If the regions originate from different partitions, and if joining them makes the graph imbalanced, the on-line repartitioning is initiated.

Repartitioning Based on the System History. Sometimes function ξ execution is unpredictable, triggered by various input changes, which makes estimates for region weights useless. Then measurements of processor loads can be used to obtain region weights, i.e. the weight w_i is proportional to cumulative time needed for execution of ξ function in region R_i.

External changes of the switch statuses and model updates (insert or delete certain elements) lead to load imbalance and cause the graph to change. In an on-line system statuses of switches are telemetered and collected from field devices. Changes of switch statuses are relatively rare. On the other hand, LF calculations are executed more frequently, which depends on other telemetered values used as inputs for the calculation. Because of such imbalance between the rate of changes of statuses of potential connections and number of calculation function executions, it is necessary to repartition regions in order to optimize use of resources, i.e. increase the speed of calculation. The importance of the research described in this paper reflects just that.

Algorithms

Four algorithms for dynamic repartitioning: DR and CP algorithms with their modifications are applied when partitions are imbalanced. Developed modifications of the algorithms are adapted to load balancing processors on NUMA multiprocessor systems [9].

Diffusion Repartitioning (DR). Diffusion algorithm design is based on Wavefront algorithm [7]. The outline of the DR algorithm pseudo code is shown below.

```
function DR_Algorithm(G_p)
Input: G_p=(V,E) – partitioned imbalanced graph
Output: balanced graph
-----------------------------------------------
for each partition $\pi_i$ in $\Pi$ do
    $outflow_{\pi i} \leftarrow CalculateOutflow(\pi_i)$
    $inflow_{\pi i} \leftarrow CalculateInflow(\pi_i)$
end for
repeat
    $\pi_S \leftarrow BestRatioPartition$(inflow / outflow), $R \leftarrow SelectRegion(\pi_S)$
    $\pi_D \leftarrow GetBestNeigbour$
    $\pi_S \leftarrow \pi_S \setminus \{R\}$, $\pi_D \leftarrow \pi_D \cup \{R\}$
    for each $\pi_X$ in $\{\pi_S, \pi_D, N(\pi_S), N(\pi_D)\}$ do
        $outflow_{\pi x} \leftarrow CalculateOutflow(\pi_x)$
        $inflow_{\pi x} \leftarrow CalculateInflow(\pi_x)$
    end for
until (G_p is imbalanced) or (no balancing progress)
while G_p is imbalanced do
    $R \leftarrow GetBestMoving$, $R \in \pi_S$ , $\pi_D \leftarrow \min(W_\pi)$
    $\pi_S \leftarrow \pi_S \setminus \{R\}$, $\pi_D \leftarrow \pi_D \cup \{R\}$
end while
```

This is an iterative process based on the calculation of two arrays, *outflow* and *inflow*. Parameter outflow_{π_i} represents the sum of the region weights that partition π_i is required to send to other partitions. It is calculated (*CalculateOutflow*) as the difference between the weight of partition W_{π_i} (3) and M $\text{outflow}_{\pi_i} = W_{\pi_i} - M$.

Parameter inflow_{π_i} represents the sum of the region weights that partition π_i is required to receive from other partitions. It is calculated (*CalculateInflow*) as the sum of *inflow* parameters of the partitions that are neighbours with partition π_i ($\pi_j \in N(\pi_i)$): $\text{inflow}_{\pi_i} = \sum_{\pi_j \in N(\pi_i)} (W_{\pi_j} - M)$.

In each iteration, a region (R) is selected (*SelectRegion*) from the partition (π_S) with the best ratio *outflow*/*inflow* (*BestRatioPartition*) and it is moved to another partition (π_D) to improve the balance and function F (*GetBestNeigbour*) [7]. If balancing is not obtained, the least connected region (R) in all overbalanced partitions (found by *GetBestMoving*) is transferred to the lowest partition.

Cut-and-Paste Repartitioning (CP). In this repartitioning [6] each border region (*BR* – set of border regions) is visited randomly. If the region is in an overbalanced partition and is a neighbour with a non-overbalanced partition, then the region will migrate to the non-overbalanced partition (only if it will not disturb the balance of the destination partition). If the region has several neighbours from different non-overbalanced partitions, then it will migrate to the partition that produces the greatest improvement in the optimization function F (*GetBestNeigbour*). After each border region is visited exactly once, the process repeats until either all partitions are balanced or no balancing progress is made. At the end, it is possible that the system is still imbalanced and then the least connected region (*GetBestMoving*) from the overbalanced partition is found and migrated to the smallest partition. The outline of the CP pseudo code is shown below.

```
function CP_Algorithm(G_p)
Input: G_p=(V,E) - partitioned imbalanced graph
Output: balanced graph
-----------------------------------------------
repeat
    for each non visited $R_b$ in $BR$ do
        $R_b \leftarrow visited$
        if ( $(R_b \in \pi_x) \wedge (W_{\pi_x} > M)$ )
            $\pi_k \leftarrow GetBestNeigbour$
            $\pi_x \leftarrow \pi_x \setminus \{R_b\}$, $\pi_k \leftarrow \pi_k \cup \{R_b\}$
            $BR \leftarrow BR \cup \{R_y \in \pi_x \mid (R_y, R_b) \in E\}$
        end if
    end for
until (G_p is balanced) or (no balancing progress)
while G_p is imbalanced do
    $R \leftarrow GetBestMoving$, $R \in \pi_S$ , $\pi_D \leftarrow min(W_\pi)$
    $\pi_S \leftarrow \pi_S \setminus \{R\}$, $\pi_D \leftarrow \pi_D \cup \{R\}$
end while
```

Modified Diffusion Repartitioning (MDR). The DR algorithm is modified to calculate *inflow* and *outflow* vectors at two hierarchical levels, which is suitable for NUMA. At the first level DR is applied to balance the sum of partition weights for all NUMA nodes, while at the second level the balance among partitions associated with a NUMA node is made. The outline of the MDR pseudo code is shown below.

```
function MDR_Algorithm(Gp)
Input: G_p=(V,E) - partitioned imbalanced graph
Output: balanced graph
```

```
numaGraph←get graph with NUMA nodes as partitions
newNUMAGraph←DRAlgorithm(numaGraph)
for each numa node in newNUMAGraph do
    subGraph←get graph of the NUMA node
    DRAlgorithm(subGraph)
end for
```

The algorithm is initiated by external changes in the state of potential connections between vertices from different partitions (nodes). If an imbalanced partition appears, the first phase of the algorithm is balancing between NUMA nodes (balancing of *numaGraph* – the graph with NUMA nodes as vertices). Therefore, two arrays are added – *outflowNode* and *inflowNode*. Parameter *outflowNode*[*i*] represents the sum of the region weights that the NUMA node *i* is required to send out to other nodes, and *inflowNode*[*i*] represents the sum of the region weights that the NUMA node *i* is required to receive from other nodes. First, parameters *inflowNode* and *outflowNode* are calculated, and they are used for deciding which regions should be moved from one NUMA node to others. After NUMA nodes are balanced, DR algorithm is applied to rearrange partitions in each node (*subGraph* – graph of partitions on a NUMA node).

Modified Cut-and-Paste Repartitioning (MCP). Introduced MCP algorithm slightly modifies CP in the process of moving regions to non-balanced partitions. The main idea of this algorithm is similar to the MDR algorithm – using repartitioning in two levels. At the first level the CP algorithm is applied only to those partitions that are placed in different NUMA nodes, while at the second level the CP algorithm is applied only to regions that are associated with the NUMA node. The outline of the MCP algorithm pseudo code is shown below.

```
function MCP_Algorithm(Gp)
Input: $G_p$=(V,E) - partitioned imbalanced graph
Output: balanced graph
numaGraph←get graph with NUMA nodes as partitions
newNUMAGraph←CPAlgorithm(numaGraph)
for each numa node in newNUMAGraph do
    subGraph←get graph of the NUMA node
    CP_Algorithm(subGraph)
end for
```

The algorithm starts when the change of the corresponding input (U_r) changes the status of potential connections. The first level deals with a graph which partitions correspond to NUMA nodes, and edges to the total weight of edges between the nodes (*numaGraph*). After that, the CP repartitioning algorithm is applied to *numaGraph*. The second phase of the algorithm involves independent repartitioning of graph related to the individual nodes (*subGraph*). In that case, the edges that connect vertices belonging to different nodes are not taken into consideration (deleting these edges appear unrelated *subGraph's*). Repartitioning algorithm of a given *subGraph* is done in the same way as a repartitioning of *numaGraph* – by applying the CP algorithm.

Experimental results

Tests were carried out regarding models of real electric power distribution network, the characteristics are shown in Table 1.

Model *it206* is a power distribution network of city Milano (Italy), while *bg5x* is five times multiplied network model of Belgrade (Serbia). In this part, test results of the dynamic repartitioning will be shown. The following experiments were carried out on a NUMA system with 2 nodes with 4 cores per node (CPU AMD Opteron 2.4GHz, 8GB RAM per core).

Table 1. Test data models

Graph name		*it206*	*bg5x*
No of elements		1787939	5980390
Initial graph (uncoarsened)	No of vertices	431102	1502505
	No of edges	434486	1523180
coarsened graph	|R|	206	315
	|Pot|	286	260
No of transformer substations		19874	30875
No of energy sources		432	320
No of transformers		18850	37625
No of line segments		42844	53270
No of switches		76387	187710

DR, CP, MDR and MCP algorithms were applied for dynamic repartitioning and tested on graphs *it206* (Tests 1 and 2) and *bgx5* (Tests 3 and 4). Tests were carried out on already partitioned graphs that were artificially made imbalanced. In all tests the imbalance is caused by changing activities of potential connections (1) between regions from different partitions. In this way, two regions become connected, making an imbalanced graph. In tests 1 and 3 new connections were made only between regions that belong to different NUMA nodes, forcing activities at the first level in MDR and MCP algorithms. On the other hand, tests 2 and 4 create new connections only between regions that belong to the same NUMA node. Each test was repeated 100 times.

Experimental results were obtained for function *F*, number of repartitioned regions (between partitions, i.e. processors), number of migrated regions (between NUMA nodes), total size of migrated regions and algorithm execution time (Table 2.). All experiments were performed using tolerance set at 10% ($\varepsilon = 0.1$).

Table 2. Repartitioning results

Test	**Alg.**	***F***	**No of repart. regions**	**No of migrated regions**	**Total size of migration**	**Time [ms]**
T1	DR	**566**	**56**	24	86722	30
	CP	**671**	78	27	57731	18
	MDR	591	60	26	28990	73
	MCP	459	96	22	**28965**	**18**
T2	DR	571	55	30	107863	32
	CP	671	78	16	36325	17
	MDR	**687**	52	0	**0**	42
	MCP	466	96	0	**0**	**13**
T3	DR	2897	14	9	81774	31
	CP	**3037**	13	6	**61016**	21
	MDR	2928	20	14	62802	64
	MCP	3001	22	12	68591	**20**
T4	DR	2897	14	6	58175	45
	CP	**3037**	13	6	43726	20
	MDR	2949	19	0	**0**	48
	MCP	3002	19	0	**0**	**11**

Experiments show that DR obtains the best results because obtained optimal value *F* is minimal in almost all tests. Total size of migrations between regions is calculated only for data transferred between NUMA nodes. Using that as a comparison criterion both MDR and MCP were giving the best results (MDR was a bit better) while DR was the worst. Additionally, the number of regions that changed partition and the number of regions moved from one NUMA node to another is shown in **Error! Reference source not found.**2, as well. It is also evident that the MCP algorithm required the highest number of regions moving between partitions, although at the same time actual number of regions migrated to another NUMA node is the smallest.

According to measured execution times the fastest algorithm is MCP, CP is a bit slower, and MDR algorithm is the slowest one (up to 4 times slower than MCP).

Taking all comparison criteria into consideration we conclude that MDR obtains better results than the other algorithms because of a smaller total migration size and good results for function *F*, although it executes slower than other algorithms. On the other hand, MCP is the fastest algorithm that obtains very good results in terms of data migration, however it is not as good for function *F* optimization as MDR is. Although, MCP provides worse results regarding function *F*, and taking into consideration that connections between partitions are potential connections, this criterion is less significant to us.

Conclusions

In this paper, we developed algorithms for dynamic repartitioning of large data model. Novel algorithms, MDR and MCP for dynamic repartitioning achieved very good results. For dynamic repartitioning we applied diffusion repartitioning (DR) and cut-and-paste (CP) algorithms. Additionally, these algorithms are modified and MDR and MCP algorithms are developed to support dynamic repartitioning running in NUMA multiprocessor systems. Experimental results prove that MDR and MCP obtain better results than DR and CP algorithms. Due to the fact that MDR algorithm is much slower in execution, we recommend using MCP algorithm for dynamic repartitioning in on-line systems. In the case when the system dynamic does not require fast algorithms, we recommend using MDR algorithm.

References

1. **Popovic M., Basicevic I., Vrtunski V.** A Task Tree Executor: New Runtime for Parallelized Legacy Software // 16th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS'2009). – San Francisco, USA, 2009.
2. **Henderson B., Kolda T. G.** Graph partitioning models for parallel computing // Parallel Computing, 2000. – No 12(26). – P. 1519–1534.
3. **Capko D., Erdeljan A., Popovic M., Svenda G.** An Optimal Relationship–Based Partitioning of Large Datasets // 14th East–European Conference on Advances in Databases and Information Systems. – Novi Sad, Serbia, 2010.
4. **Cybenko G.** Dynamic load balancing for distributed memory multiprocessors // Journal of Parallel and Distributed Computing, 1989. – No. 2(7). – P. 279–301.
5. **Hu Y.F., Blake R. J.** An optimal dynamic load balancing algorithm. – Technical Report DL–P–95–011. – Daresbury Laboratory, Warrington, UK, 1995.
6. **Schloegel K., Karypis G., Kumar V.** Multilevel diffusion schemes for repartitioning of adaptive meshes // Journal of Parallel and Distributed Computing, 1997. – No. 2(47). – P. 109–124.
7. **Schloegel K., Karypis G., Kumar V.** Wavefront diffusion and LMSR: Algorithms for dynamic repartitioning of adaptive meshes. – Technical Report TR 98–034. – Dept. of Computer Science and Engineering, University of Minnesota, 1998.
8. **Meyerhenke H.** Dynamic Load Balancing for Parallel Numerical Simulations based on Repartitioning with Disturbed Diffusion // 15th International Conference on Parallel and Distributed Systems. – IEEE Computer Society, 2009. – P. 150–157.
9. **Correa M., Chanin R., Sales A., Scheer R., Zorzo A. F.** Multilevel Load Balancing in NUMA Computers. –Technical Report TR 049. – PUCRS, Porto Alegre, 2005.
10. **Energy management system application program interface.** IEC 61970, EMS–API. – Part 301: Common Information Model (CIM) Base. – IEC, 2007.
11. **Vukmirović S., Erdeljan A., Lendak I., Čapko D.** Extension of the Common Information Model with Virtual Meter // Electronics and Electrical Engineering. – Kaunas: Technologija, 2011. – No. 1(107). – P. 59–64.
12. **Shirmohammadi D. Hong H., Semlyen A., Luo G.** A Compensation–Based Power Flow Method For Weakly Meshed Distribution And Transmission Networks // IEEE Trans. on Power Systems, 1988. – No. 2(3).– P. 753–762.

Received 2011 12 13
Accepted after revision 2012 01 14

D. Capko, A. Erdeljan, G. Svenda, M. Popovic. Dynamic Repartitioning of Large Data Model in Distribution Management Systems // Electronics and Electrical Engineering. – Kaunas: Technologija, 2012. – No. 4(120). – P. 83–88.

In this paper, modern Distribution Management Systems (DMS) that utilize multiprocessor systems for efficient processing of large data model are considered. The aim of the research is to obtain an optimal load balancing among processors in terms of memory usage and the calculation execution time. The dynamic repartitioning is performed during execution when an imbalance is detected. Diffusion Repartitioning (DR) and Cut-Paste (CP) algorithms for dynamic repartitioning are discussed. Furthermore, modified versions of DR and CP algorithms, named MDR and MCP, are developed in order to improve dynamic repartitioning running in Non-Uniform Memory Architecture (NUMA) multiprocessor systems. The proposed algorithms were applied on data model describing large power distribution network. Experimental results prove reductions of processors' load imbalance and performance improvements. Bibl. 12, tabl. 2 (in English; abstracts in English and Lithuanian).

D. Capko, A. Erdeljan, G. Svenda, M. Popovic. Didelio duomenų kiekio modelio taikymas paskirstymo ir valdymo sistemose // Elektronika ir elektrotechnika. – Kaunas: Technologija, 2012. – Nr. 4(120). – P. 83–88.

Nagrinėjamos modernios pasiskirstymo valdymo sistemos, naudojančios multiprocesorines sistemas didelio duomenų kiekio modeliui efektyviai apdoroti. Šio tyrimo tikslas – gauti optimalią procesorių apkrovą atminties panaudojimo ir skaičiavimo trukmės atžvilgiu. Vykdymo metu atliekamas dinaminis perskirstymas, kai aptinkamas išsibalansavimas. Aptarti difuzinio perskirstymo ir Cut-Paste algoritmai. Be to, atliktas minėtų algoritmų modifikavimas siekiant pagerinti dinaminį perskirstymą, atliekamą multiprocesorinėse sistemose. Pasiūlyti algoritmai buvo pritaikyti didelio galios tinklo modelio duomenims. Eksperimentai parodė procesorių apkrovos išsibalansavimo sumažėjimą ir našumo padidėjimą. Bibl. 12, lent. 2 (anglų kalba; santraukos anglų ir lietuvių k.).