[Downloaded from www.aece.ro on Saturday, October 13, 2018 at 09:23:56 (UTC) by 159.226.100.198. Redistribution subject to AECE license or copyright.]

Load Balancing of Large Distribution Network Model Calculations

Lajos MARTINOVIC[1,2], Darko CAPKO[1], Aleksandar ERDELJAN[1]
[1]*Faculty of Technical Sciences, University of Novi Sad, 21000, Serbia*
[2]*Schneider Electric DMS NS, 21000, Serbia*
dcapko@uns.ac.rs

***Abstract*—Performance measurement and evaluation study of calculations based on load flow analysis in power distribution network is presented. The focus is on the choice of load index as it is the basic input for efficient dynamic load balancing. The basic description of problem along with the proposed architecture is given. Different server resources are inspected and analyzed while running calculations, and based on this investigation, recommendations regarding the choice of load index are made. Short description of used static and dynamic load balancing algorithms is given and the proposition of load index choice is supported by tests run on large real-world power distribution network models.**

***Index Terms*—distributed computing, power distribution, power system analysis computing, power system management, smart grids.**

I. Introduction

Power distribution networks are part of the electric networks connecting the transmission networks and the end consumers, thus, supplying the end users with electricity [1]. To deliver electric power more efficiently, reliably and more securely, smart grids with modern metering devices and communication infrastructure are developed. The real-time monitoring of the medium and low voltage distribution networks is restricted mostly due to economic and technical limitations. Real-time measurements are typically located on primary substations at 33/11kV and sparsely on lower voltages i.e. on 11kV and on 380V. With the growing number of *DER*s (distributed energy resources) such as distributed generators, home batteries and electric vehicles with demand response programs the number of measurements has proven to be insufficient. Therefore, in contrast to state estimation in electric power transmission networks where it is usually used to filter out bad measurements, in distribution networks state estimation has a crucial part, to determine the state of power network from a limited number of measurements [2]. To meet the demands of the consumers, along with the smart grid development, it is necessary to be able to monitor distribution networks and perform real-time decisions when power outages happen. The number of consumers is usually very large, and additionally, power distribution networks are built using various equipment, resulting in substantial network size, and complexity. These factors and requirements have evolved into the development of *Distribution Management Systems* (*DMS*). As such, one of the main expectations of the *DMS* software is the execution of network analysis function, including the fast load flow, state estimation, contingency analysis, optimal capacitor placement etc. Since electric power distribution companies are managing large data models the scalability of the *DMS* software is also an essential factor.

Scalability of a software describes the ability to handle increasing or decreasing workload whether by vertical or by horizontal scaling. Vertical scalability is achieved by improving the computing power with faster processors more memory, faster hard disk etc. Horizontal scalability on the other hand is accomplished by adding more servers, and it has obvious advantages: smaller costs, virtually no limitation, easier upgradeability, improved resilience of the system etc. In order to utilize horizontal scaling, the software must be able to effectively parallelize tasks. This can be done on several levels, by distributing tasks to separate computers, by distributing tasks on separate CPUs within the same computer, or by separating tasks to different threads on the same CPU. In related works [3-4] the focus is on parallelization of the power analysis algorithm itself. In this paper, a different approach is explored: parallelization of power analysis related calculations by dividing the distribution power network up-front into smaller parts.

Distribution networks are usually weakly meshed with radial topology [1]. This radial topology configuration gives the possibility to divide these large distribution network models into smaller parts, to so-called *roots*, and perform calculations based on load flow analysis [5] independently [6]. Distributed computer networks and distributed systems with efficient load balancing are a good choice for the previously addressed real-time response problems. Load balancing can be described as a method for distributing the workload to different computational nodes in effort to minimize response time and/or distribute the workload evenly between available nodes. The division of the power distribution network into independent parts i.e. roots, gives the possibility to distribute the roots data and calculations related to them to different computational nodes [7]. It is shown in [8] that the initial workload can be estimated ahead based on the configuration of the network. Different equipment types (non-zero-impedance branches, generators, consumers etc.) have been considered, and a fairly good initial load distribution can be done based on the quantity of different types of equipment.

The aim of this paper is to select good performance indices for load balancing distributed systems in order to

This work was supported in part by Schneider Electric DMS NS, by providing real world distribution network model data for testing and validating the results. This work has been partly supported by the Serbian Ministry of Education, Science and Technological Development, through grants No. 32018.

Digital Object Identifier 10.4316/AECE.2017.04002

[Downloaded from www.aece.ro on Saturday, October 13, 2018 at 09:23:56 (UTC) by 159.226.100.198. Redistribution subject to AECE license or copyright.]

achieve fast power systems calculations. The focus is on selection of proper measure for measuring the workload. The selection of the right performance indicators is critical for the balancing of the workload and for sustaining system stability within DMS with large power distribution network. The concepts used for workload balancing on system startup and for maintaining it during system uptime as well are presented in following sections and supported with experimental results in Section IV.

II. Architecture

A. Functional architecture overview

The proposed centralized architecture of the system from the aspect of functionality and responsibility is illustrated in Fig. 1.

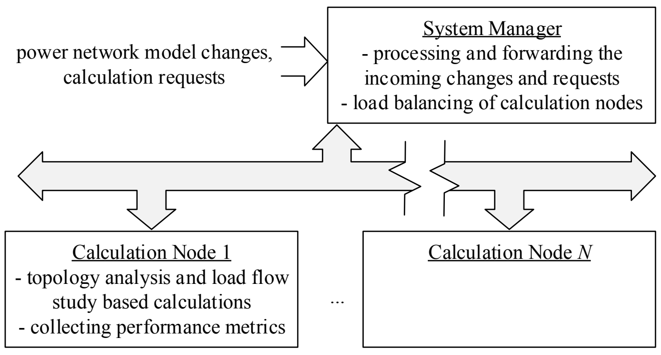
power network model changes, calculation requests
System Manager
- processing and forwarding the incoming changes and requests
- load balancing of calculation nodes
Calculation Node 1
- topology analysis and load flow study based calculations
- collecting performance metrics
...
Calculation Node N

Figure 1. Functional system overview

The *System Manager* is responsible for tasks such as receiving, processing and forwarding the incoming changes in the power distribution network model, forwarding the on-demand calculations requests and the load balancing. The *System Manager* is connected via bus to every *Calculation Node* (*CN*) which are located on individual machines in a computer cluster. The *CN*s are dedicated to running topology analysis and calculations which are based on load flow analysis on a part of the power distribution network, which has been assigned by *System Manager* to that specific *CN*.

B. Components overview

The *System Manager* includes two components: *Model Manager* and *Load Balancer*, as illustrated on Fig. 2. At the beginning, the initial computational load is distributed by assigning different parts of the power distribution network to every *CN*. The initial power network partitioning to the so-called roots is done based on the networks configuration [6], which gives the starting workload distribution. The Model Manager maintains information about which part of the power network is assigned to which *CN*. Calculations are performed when change in the network topology configuration is detected or when an on-demand calculation request is received. The *Model Manager* forwards the incoming changes and requests to correspondent *CN*. Consequently, the *CN* processes and stores the changes in a *Network Model*, which performs topology analysis in a *Topology Analyzer* (*TA*) and runs calculations in a *Function Calculator* (*FC*). The modules on *CN*s are running on different processor cores.

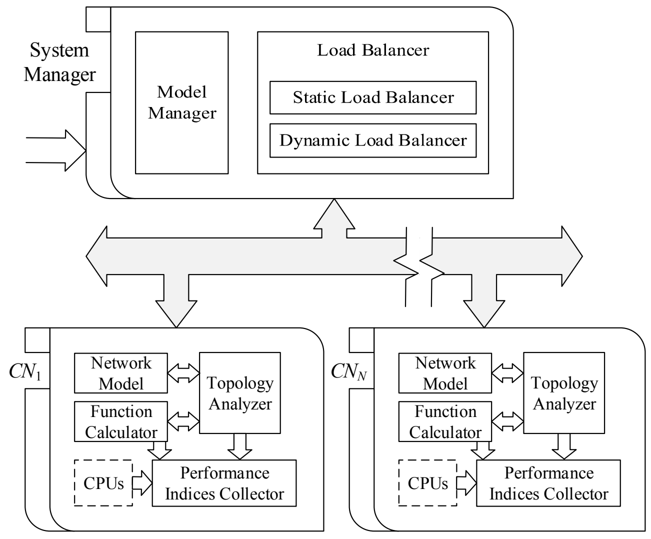

Figure 2. Architecture of the system

The *Load Balancer* is liable for the performance monitoring and workload balancing between the *CN*s. During runtime, the *Performance Indices Collector* components collect the selected performance metrics from every node. The *Load Balancer* is then periodically invoked, and inspects the state of the system and the need to redistribute the workload between *CN*s based on gathered workload information.

III. Load Balancing

The essential problem of load balancing and different algorithms for achieving it, are well studied and covered in literature [9-11]. Tersely, the main objectives of load balancing are the minimization of the response time, optimization of the systems resource usage and preventing the overload of any resource. From the aspect of which available information is used for workload distribution and whether it is done prior to or during runtime, load balancing strategies can be classified in two major categories: static and dynamic. As such, static load balancing is based only on available knowledge about the system before runtime. Adversely, dynamic load balancing implies using information about the current state of the system when making workload re-distribution decisions. This also entails that static load balancing is simpler, easier to implement and has coherent execution time, however using dynamic load balancing results in a better workload distribution. Therefore, the *Load Balancer* component has two parts, the static and dynamic load balancers.

A. The Static Load Balancer

Static load balancing is used only at start-up when the power distribution network is being divided into roots based on normal switch statuses. The root weights are calculated based on the networks configuration, and roots are assigned to *CN*s. At this stage, neither communication costs i.e. the amount of data exchanged between *CN*s and the *System Manager* nor the rate of calculation requests are known.

A1. Optimization problem

Static load balancing can be viewed as a graph partitioning problem [8], but since tie switches connecting

[Downloaded from www.aece.ro on Saturday, October 13, 2018 at 09:23:56 (UTC) by 159.226.100.198. Redistribution subject to AECE license or copyright.]

different roots are off – the problem is reduced to a well-known, *NP*-complete set partitioning problem. Given the finite set of numbers A ⊂ ℤ+, partition A to such subsets An that the difference between the sum of each subset is minimized. Translated to the current problem: given the power distribution network divided to roots and with computational weight of each root wi, find such sets of roots that the sum of the roots weight on each *CN*s are equal as possible. The weight of the entire distribution network can be written as a set R = {w1,...,wr}, where r is the number of roots. With given number of calculation nodes N, find such subsets Rn ⊂ R for n = 1…N, so that difference of the subset sum is minimized

$$D = \max\left(\sum_{w_i \in R_n} w_i\right) - \min\left(\sum_{w_i \in R_n} w_i\right) \tag{1}$$

under conditions that the subsets are:

- mutually exclusive Ri ∩ Rj = ∅ for i≠j, j,i = 1…N
- collectively exhaustive R1 ∪ R2 ∪ … ∪ Rn = R.

The brute-force search for all possible subsets would have exponential runtime, O(2r), so it is only applicable to small sets. Since the workload distribution will be later refined with dynamic load balancing, finding the optimal solution is not crucial at start-up. Few algorithms were considered for static load balancing for which short previews are given below.

A2. Algorithms

The *greedy heuristic* algorithm (*GH*) is straightforward. Starting with sorted root weights in decreasing order, move each root weight to a set with the least sum until no weights are left [12]. It requires set sorting, which has $O(r \log r)$ complexity and root assigning with $O(r)$ complexity.

The *set differencing method* [13], also known as *Karmarkar-Karp* heuristics algorithm (*KK*), as the previous algorithm, starts with an ordered set of root weights. For every element in the set an *N*-tuple is created, and the corresponding root weight is initially assigned as the first element of tuple. At every iteration of the algorithm, the 2*N* largest elements from two *N*-tuples $(a_1,\dots,a_N)$ and $(b_1,\dots,b_N)$ are combined according to the rule

$$C = (a_1 + b_N, a_2 + b_{N-1}, \dots, a_N + b_1) \tag{2}$$

The previously combined tuples are removed, and the new tuple *C* is normalized by subtracting the smallest value found in it. The tuple is then sorted in descending order, and placed back into the sorted set of *N*-tuples. This is repeated until only one tuple is left. Finally, the last tuple represents the difference of the resulting subsets sums. To retrieve the actual subsets additional accounting is necessary while combining the tuples. The *KK* algorithm runs in $O(n \log n)$ time.

The *sequential number partitioning* (*SNP*) [14] first runs the *KK* algorithm to retrieve the initial upper and lower bounds. The upper bound u_b is the best partition i.e. the sum of set divided with the number of partitions

$$u_b = \frac{1}{N}\sum_{i=1}^{r} w_i \tag{3}$$

and the lower bound l_b is calculated as

$$l_b = \frac{1}{N}\left(\sum_{i=1}^{r} w_i (N-1)\cdot d\right) \tag{4}$$

where *d* is the difference between the best *N*-way partition found by *KK*.

Next, a binary exclusion-inclusion tree is built with every level in the binary tree representing a root weight from the set. The leaves correspond to subsets including the root weights on left branches and excluding on right. The inclusion tree is then searched for subsets which satisfy the conditions that the subset sum at the leaf node is less than u_b, and the subset sum at the leaf node plus the sum of non-assigned root weights is larger than the l_b. For the remaining partition, *SNP* is then called recursively with l_b greater or equal than previous subset sum. The worst-case complexity of tree search is $O(2^r)$.

In future works, next to the presented and implemented algorithms, the principles and methodologies presented in [15-18] could be considered as well, where alongside nature inspired algorithms better solution is obtained using hybrid algorithms.

B. Workload estimation

Since one of the key issues with good dynamic load balancing is recognizing an adequate load index, the choice is explained in more detail in the following section. The load index should represent the workload on some node. Therefore, the following questions arise: how to estimate the workload on a *CN*, which performance indicators should be collected and used as load index? With some exceptions [19-21], mainly configurations with homogenous nodes are analyzed. In [22-23] the authors used various load indices, but they all have similar reasoning regarding the selection of parameters for load index. The incoming job, or in this case the load flow analysis, will demand some resources from the *CN* i.e. I/O queues, CPU and memory consumption, and therefore, the CPU queue length, the CPU utilization, the normalized response time, etc. will reflect the workload on the node. The authors have been using one of the mentioned per, or some simple combination of them. It is shown in [24-25] that the CPU queue gives a good estimate of the workload and it can be used as load index. In some situations, the CPU queue does not reflect the real workload on the *CN*. For example, if some *CN* has a large hard disk I/O, the read/write operations will be blocked until served, and thus the hard disk will be a bottleneck and CPU will be idle. Fortunately, this can be measured with disk queue i.e. the number of request waiting for the disk.

B1. Performance metrics

With the centralized architecture, presented in Section II, various system performance metrics were taken into consideration: CPU usage and CPU queues, memory loads, disk queues and transfer, etc. As the model of the power network is held in the *CN*s memory, there is low or none disk I/O, and thus the performance indicators related to the hard disk are dismissed at the start. For the same reason, the memory consumption is not descriptive enough either, since the small fluctuations in memory usage are just the result of the process of creating the calculation model for load flow analysis itself. The remaining performance indicators, the CPU usage and the CPU queue length, have shown as good estimate of the workload.

Since *TA* and *FC* are working in tandem, it has been observed that frequent calculation requests can result in the overload of one of the components (depending on the type

[Downloaded from www.aece.ro on Saturday, October 13, 2018 at 09:23:56 (UTC) by 159.226.100.198. Redistribution subject to AECE license or copyright.]

of calculation), leaving the other components inactive. In these situations, where *TA* and *FC* modules are running on different processor cores, the total CPU usage will not reflect the real workload, since one processor core will be idle. To tackle this problem, additional performance counters are introduced: the length of the request queues on *TA* and on *FC*. Thus, the load index of the *j*-th node i.e. CN_j can be expressed as some function of the collected performance indices.

Considering the above, from all possible performance metrics the following subset is chosen:

$$\mathcal{P}_M=\{C, Q_T, Q_F\} \tag{5}$$

where C, Q_T, Q_F denote total processor time, *TA* and *FC* request queue lengths, respectively, for the observed CN_j.

B2. Load index

The load index L_j for CN_j will be some function of collected values defined by variables from $\mathcal{P}_M$

$$L_j = f(P) \tag{6}$$

with P defined as

$$P = \begin{bmatrix} P_1 & P_2 & \dots & P_n \end{bmatrix} \tag{7}$$

representing the vector of collected performance index triplets for n samples i.e. the i-th sample on the observed CN_j will be

$$P_i = \begin{bmatrix} C & Q_T & Q_F \end{bmatrix}^{\mathrm{T}} \tag{8}$$

A simple linear combination of these variables yields a good workload estimate. Let's define the function f from (6) as a mapping of P to the interval [0, 1]

$$f : P \mapsto [0,1] => L \in [0,1] \tag{9}$$

where an unloaded node will have load index 0 and overloaded node will have 1. Subsequently, define l_i as a vector of combined performance indices $C\ Q_T\ Q_F$

$$l_i = K \cdot P_i \tag{10}$$

for each node $i = 1\dots n$ and with weight factors K defined

$$K = \begin{bmatrix} K_1 & K_2 & K_3 \end{bmatrix} = \frac{1}{m}\begin{bmatrix} \frac{K_C}{100} & \frac{K_T}{r_j} & \frac{K_F}{r_j} \end{bmatrix} \tag{11}$$

where r_j is the number of roots assigned to CN_j, and $m=|\mathcal{P}_M|=3$.The coefficients K_C, K_T and K_F, are arbitrary chosen values allowing to increase or decrease the impact of some performance metric variables, with constraint

$$\sum_{k=1}^{m} K_k = 1\,. \tag{12}$$

For the value of one for the coefficients K_C, K_T and K_F, every performance index will have the same impact on the resulting load index.

CPU usage is a frequently changing variable, and load flow analysis also has a short execution time, hence some smoothing function is necessary. To flatten these short-term peaks exponential smoothing is used over the entire time of the observation. The smoothing function is applied as

$$\hat{l}_i = \alpha \cdot l_i + (1-\alpha)\cdot \hat{l}_{i-1} \tag{13}$$

with smoothing factor $\alpha = [0,1]$ as by definition

$$\alpha = 1 - e^{-\Delta t/\tau} \tag{14}$$

where Δt is the sampling period of the collected performance indices and τ is the time constant i.e. the elapsed time of a step response needed to reach 63.2% of its original value. For smaller α, the older values will have more impact and the function will be "smoother". The initial value in exponential smoothing (13) is set to the first value of load index i.e.

$$\hat{l}_1 = l_1\,. \tag{15}$$

Considering that DMS is a mission-critical application, threshold limit of 75% is imposed to CPU usage, and with the same reasoning, thresholds for queue lengths are defined as the number of roots r_j, assigned to CN_j. Finally, the load index for CN_j will have the form (16).

$$L_j = \begin{cases} \hat{l}_i & \text{for } (C_{(n)} \le 75) \wedge (Q_{T(n)} \le r_j) \wedge (Q_{F(n)} \le r_j) \\ 1 & \text{otherwise} \end{cases} \tag{16}$$

The workload state of the whole system can be described with a set of load index (16) of every *CN*.

$$L = \{L_1, L_2,\dots, L_N\} \tag{17}$$

C. *The Dynamic Load Balancer*

The main motivation for dynamic load balancing is the improvement of the workload distribution by using information about the current state of the system.

Three main parts of the dynamic load balancing can be observed [9]:

- local workload information of *CN*
- information policy
- transfer policy.

The *information policy* describes the procedure of sharing the information in the system (between which components and in which manner is information broadcasted, how frequently, etc.). Since the architecture of the system is centralized, all *CN*s are sending their state to *Load Balancer* at *System Manager* periodically.

The *transfer policy* decides when is it advantageous to migrate the task and to which *CN* it should be transferred.

The *workload information* represents the current workload state of the *CNs*.

C1. Optimization problem

After an arbitrary period, based on the collected load indices, presented in the previous section, if imbalance is detected, that is, when the difference between load indices defined with (16) is larger than the predefined limit, the workload should be redistributed. From *CN*s with high workloads some roots (together with calculations), are migrated to other *CN* with less workload. The aim is to minimize the imbalance i.e. to minimize the cost function defined with mean squared error

$$J = \min \sum_{j=1}^{N} \left(L_j - \overline{L}\right)^2 \tag{18}$$

where $\overline{L}$ is the average value of all load indexes

$$\overline{L} = \frac{1}{N}\sum_{j=1}^{N} L_j \tag{19}$$

Since the size of the model for one root can be significant, the migration of the model data can be expensive, i.e. the communication costs cannot be neglected and they should be considered as well. Therefore, the dynamic load balancing should not be triggered below a certain threshold value. Like with static load balancing, several algorithms are tested. Given the fact that the initial workload distribution is already done with static load balancing, it is in interest to consider algorithms for dynamic load balancing which start

[Downloaded from www.aece.ro on Saturday, October 13, 2018 at 09:23:56 (UTC) by 159.226.100.198. Redistribution subject to AECE license or copyright.]

with already distributed workloads, and perform refinement in the form of reallocating just some of the roots to other nodes, so that the migration cost is minimized. The aim of the dynamic load balancing algorithm is the minimization of the function

$$F = F_m + D \qquad (20)$$

where F_m is the total migration cost between *CN*s, and D is defined with (1).

Under the assumption that the network connection is the same between the *CN*s, the migration cost is reduced to the size of the computational model of the migrated roots.

$$F_m = \sum_{s_i \in M} s_i \qquad (21)$$

where s_i is the size of the computational model of *i*-th root, and M is the set of all migrated roots.

In current application, the sizes of the computation models, i.e. the migration costs of the smallest roots are up to MB while the largest roots can have computational model of few hundreds of MB.

C2. Algorithms

One of the diffusion based method for graph partitioning is the *Cut-Paste* (*CP*) [26]. In every iteration, for each overloaded *CN* (donor), a root is chosen and migrated to an underloaded *CN* (receiver) so that the resulting imbalance is minimized with least possible migration i.e. the optimization function (21) is minimized.

A very similar technique, the *Match-Maker* (*MM*) is presented in [27]. The most loaded *CN* (source) is paired with the least loaded *CN* (destination), the second most loaded *CN* is paired with second least loaded *CN* and so on. Then, roots are selected from sources and migrated to destination. The selection is simple: find the largest root in source that can be migrated to destination, so that the it will not become overloaded.

In both *CP* and *MM* algorithms, the process is repeated until there is no more improvement or until the desired value of workload balance is reached.

Another group of algorithms for repartitioning is the so-called scratch-remap algorithms. While the diffusion based algorithms start with already distributed workloads and attempt to achieve better balancing between the *CN*s, the scratch-remap algorithms calculate balanced workload distributions from scratch, and then try to minimize the difference between the existing and the calculated workload distributions. One of the scratch-remap algorithms is the *Locally-Matched Multilevel Scratch-Remap* (*LMSR*) presented in [28]. *LMSR* can be described in three main steps:

1. Determining the new balanced workload distribution - the balanced partitioning is obtained using algorithms for partitioning from scratch.
2. Similarity matrix – in this step a similarity matrix is constructed. The similarity matrix *S* is a square matrix of size *N* (where *N* is the number of *CN*s). Each cell of that matrix represents the measure of similarity between the existing imbalanced and the newly calculated balanced load distribution from 1st step. As the goal is to minimize the migration between nodes while reaching balanced workload distribution, the root sizes are of adequate measure, and hence the element $S(i, j)$ is the sum of weights of overlapping roots that belongs both to CN_i in imbalanced and to CN_j in the newly calculated balanced workload distribution.
3. Mapping - after the similarity matrix is constructed, distinct pairs of unbalanced and balanced *CN*s are chosen in effort to minimize the migration cost. From each row and column of the matrix *S* one cell must be chosen so that their sum is the maximum possible. This can be reduced to a well know assignment problem or finding a maximum weight matching in a weighted bipartite graph, which can be solved using the *Hungarian* algorithm in $O(n^2)$ running time.

IV. Results

To justify the choice of performance metrics and to prove the validity of the (17) experimental tests were conducted. which included triggering load flow analysis and state estimations on real power distribution networks using *CN*s with similar characteristics.

A. Experimental setup

The two main components on each node, the *TA* and *FC* modules, described in section II, were assigned to different processor core. The calculations were triggered in different time intervals on various roots. The frequency of the calculation requests was depending on root size, and the performance indicators defined in (5) were collected. The CPU and the queue lengths are sampled on every second. All tests were performed using a cluster of PCs with similar characteristic (Intel i5 processors and 32GB RAM memory). The iterative algorithm for load flow analyses is written in Intel Fortran programming language, while the topology analysis part is written in C++.

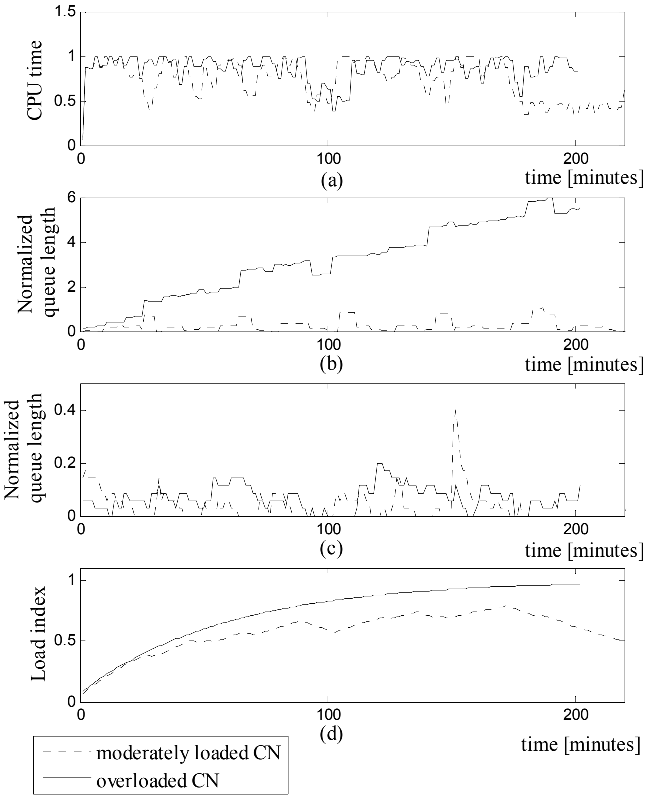

Figure 3. Normalized CPU usage (a), *TA* (b) and *FC* (c) queue lengths and load index (d) for overloaded and for moderately loaded *CN*

[Downloaded from www.aece.ro on Saturday, October 13, 2018 at 09:23:56 (UTC) by 159.226.100.198. Redistribution subject to AECE license or copyright.]

Two typical example of the collected data is illustrated in Fig. 3. where the normalized performance samples of CPU and queue lengths for one node are shown. There was no emphasis on neither of the performance index and therefore the coefficients K_C, K_T and K_F defined with (11) are set to the value of one. In this example, the collected normalized values $C/100$, (Fig 3. a), Q_T/r_j (Fig 3. b) and Q_F/r_j (Fig 3. c) and the resulting load index (Fig 3. d), which varies between 0.5-0.75, shows that the *CN* is mildly loaded (shown as *moderately loaded CN*). With more frequent calculation requests, the same variables are presented on the same Fig. 3 (shown as *overloaded CN*). It can be concluded, that the *TA* module in this setup is not capable of performing the topology analysis at a given rate, thus resulting in pileup of requests on the queue Q_T as shown in Fig. 3. b, while the queue length on the *FC* module (Fig. 3 c) and the CPU is in normal range. The resulting load index (16) in this case approaches the value of one.

Different configurations and combinations of above described static and dynamic load balancing algorithms were tested. The complete tests i.e. static load balancing, collecting the performance indices and the dynamic load balancing is performed for a period of few weeks. The network model is taken from a real power distribution networks, both European and North American. Some insignificant differences have been observed in load balancing due to different network configuration, voltages, and since North American distribution networks are mostly unbalanced. The more detailed description of the used data model can be found in [8].

The representative example shown in the following figures illustrates the obtained results for power distribution network consisting from approximatively 400 roots of different sizes, which are distributed to five homogenous *CN*s.

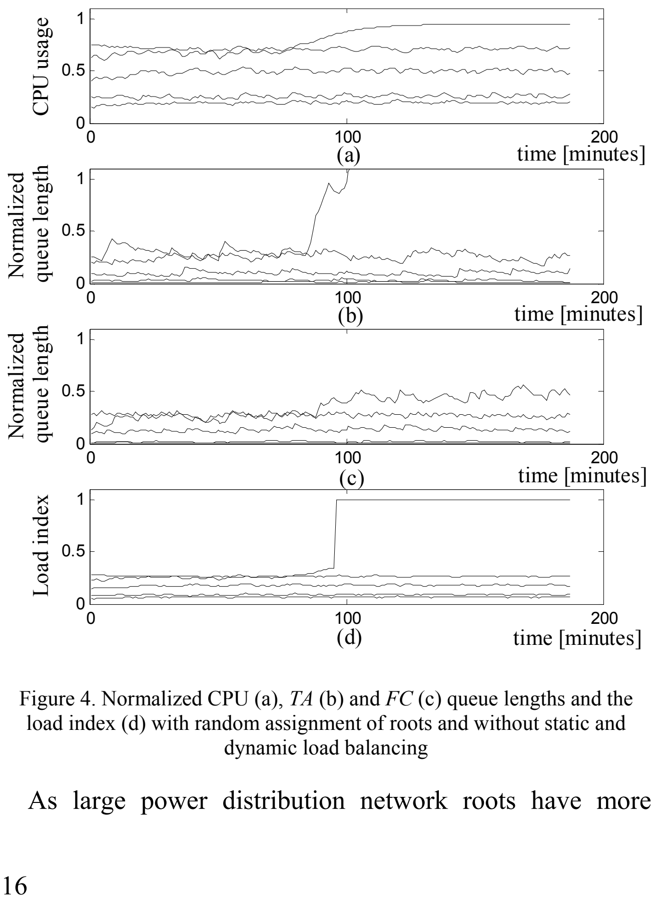

Figure 4. Normalized CPU (a), *TA* (b) and *FC* (c) queue lengths and the load index (d) with random assignment of roots and without static and dynamic load balancing

As large power distribution network roots have more switching devices there is a much higher probability that calculations will be requested for these roots due to network reconfiguration caused by operating with switching devices. This can be concluded from collected statistic values from field as well.

B. Experimental results

With random assignment of the roots to the *CN*s i.e. without static and dynamic load balancing the collected data are shown in Fig. 4. It can be observed that the workload is not evenly distributed between nodes, the CPU usage on some nodes is very low. After a while the normalized *TA* queue length on one node starts increasing up to the point where the component is not capable of handling any more the incoming requests at this rate. Since the frequency rates of the calculation requests are not known upfront, they are not considered in static load balancing and the initial workload distribution is very inaccurate.

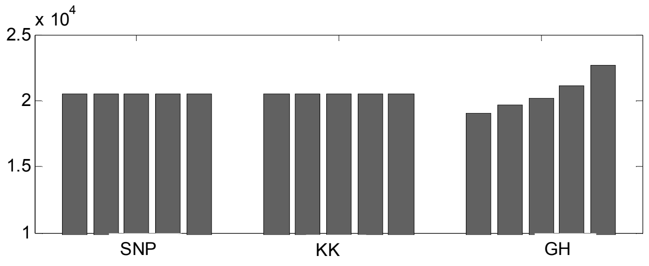

Figure 5. Comparison of the workload distribution after static load balancing: *Sequential Number Partitioning* (*SNP*), *Karmarkar-Karp* (*KK*) and *Greedy Heuristic* (*GH*)

The comparison of initial imbalance on *CN*s after partitioning with *SNP*, *KK* and *GH* is shown in Fig. 5. Both *SNP* and *KK* found nearly perfect distribution while the *GH* resulted in somewhat larger workload imbalance.

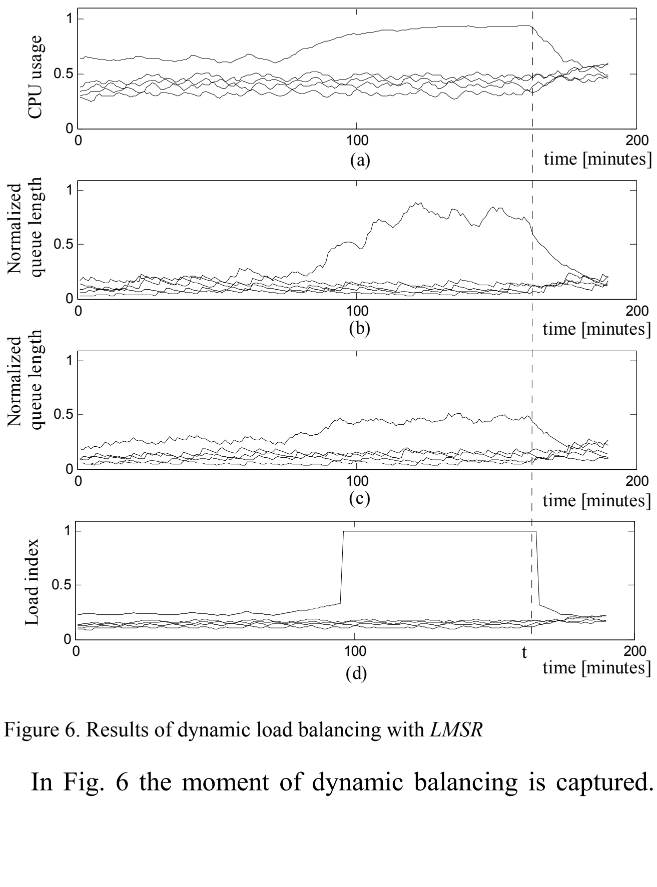

Figure 6. Results of dynamic load balancing with *LMSR*

In Fig. 6 the moment of dynamic balancing is captured.

[Downloaded from www.aece.ro on Saturday, October 13, 2018 at 09:23:56 (UTC) by 159.226.100.198. Redistribution subject to AECE license or copyright.]

After a while the load index (defined with (16)) has taken the value of one. Since it is not desired to perform load balancing immediately (the imbalance can be just a result of some short-term anomaly e.g. power outage), some delay has been added. At time *t,* after the load index was high for given time (60 minutes in this setup), the dynamic repartitioning is triggered.

The weights of roots are recalculated using previous weights and the rate of requests. With the newly obtained weights, repartitioning is performed and some roots, determined by the dynamic load balancing strategy, are migrated. On the example shown on Fig. 6, *LMSR* algorithm is used for dynamic load balancing. After migration, the workload on the previously overloaded *CN* dropped and the performance measurements are showing that the workloads are fairly distributed to all *CN*s. Along with the improvement of the load index (Fig. 6 d), the CPU utilization and the request queue lengths are uniformly distributed (Fig. 6 a-c).

The *LMSR* algorithm, as expected, has resulted in a better workload redistribution which is illustrated in Fig. 7 a, however the data migration is very large comparing to *MM* and *CP*. In the current setup, the size of the computational model for one root is ranging from few MB up to few hundreds of MB and therefore the migration cost cannot be neglected. When the best possible workload distribution is not mandatory, i.e. some small arbitrary imbalance is allowed, the *CP* and the *MM* algorithms can be easily modified to stop when the desired value of workload imbalance is reached.

Because of the nature of diffusion algorithms (*CP* and *MM*) the iterative improvement process can be easily stopped which is not the case with remapping from the scratch (*LMSR*). This has resulted in very small increase of the overall workload imbalance (Fig. 7 b), and almost unnoticeable change in the load index, but the reduction of data migration is significant.

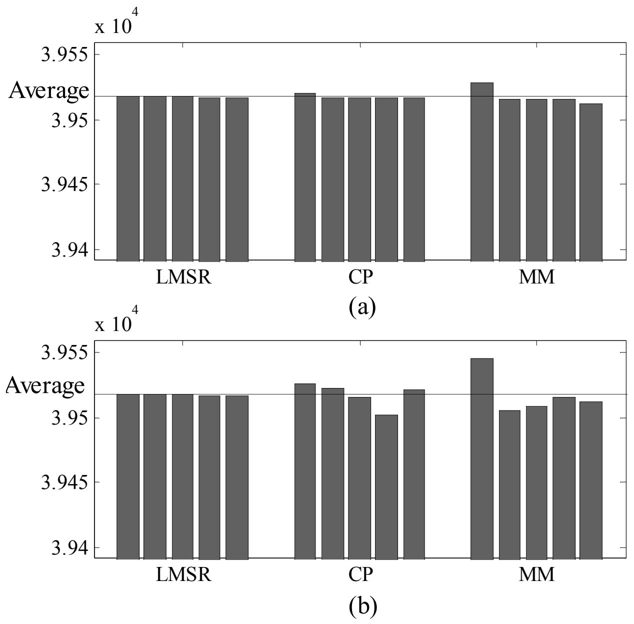

Figure 7. Workloads on the calculation nodes after dynamic load balancing (a) and after dynamic load balancing with allowed imbalance (b)

In Table I, along with the cost function (18), the size of the migrated data is shown when the best workload balancing is obtained and when this requirement is relaxed i.e. small imbalance is allowed in *CP* and in *MM*. Results show that the migration cost can be significantly reduced with *CP* or *MM* algorithm when the best workload balance is not mandatory. Based on the obtained result it can be concluded that by using the mentioned algorithms for static and dynamic load balancing the workload balance can be improved. The best workload balance is achieved with LMSR algorithm, which is advantageous in cases when all *CN*s have high workload. In those scenarios, even small imbalance can lead to overload of one of the *CN*s which will lead to instable system. On the other hand, the *CP* algorithm has shown to be better solution compared to the *MM* algorithm both in terms of workload balancing, and migration cost. Comparing to the *LMSR* algorithm the *CP* algorithm performed with somewhat worse workload balancing but gave a much smaller migration costs.

TABLE I. COST FUNCTION AND MIGRATION COST

	Before	*LMSR*	*CP*	*MM*
	Best solution that can be found			
J	0.585	0.003	0.015	0.068
Data migration [MB]	-	8671	1694	2093
	With small imbalance allowed			
J	0.585	0.003	0.02	0.083
Data migration [MB]	-	8671	1493	2052

V. CONCLUSION

This paper proposes the selection of the load index and comparing static and dynamic load balancing strategies for distributing calculations to calculation nodes are presented. The solution is applied to power system analysis calculations where methodology and arguments behind the selection of performance indices are given. Based on performance measurements and on the characteristics of load flow analysis on weakly meshed power distribution networks the choice of load index is elaborated along with short overview of different load balancing strategies is given. The conjecture given in section III is proven with test results. All static load balancing methods gave reasonably good results with *GH* performing slightly worse than *KK* and *SNP*. From dynamic load balancing strategies considered in this work *LMSR* gave the best workload balancing, however when small imbalance can be tolerated *CP* outperformed *LMSR* and *MM* in terms of speed of execution and size of migrated data.

ACKNOWLEDGMENT

We would like to thank our associates at Schneider Electric DMS NS, Novi Sad, Serbia for their help during our work on this research. They were the sole source of all real-life electric power system data.

REFERENCES

[1] Lakervi and Holmes, Electricity Distribution Network Design. Institution of Engineering and Technology, pp 1-26, pp. 192-208 2003. doi:10.1049/PBPO021E

[2] R. Singh, B. C. Pal, and R. B. Vinter, "Measurement Placement in Distribution System State Estimation," IEEE Transactions on Power

[Downloaded from www.aece.ro on Saturday, October 13, 2018 at 09:23:56 (UTC) by 159.226.100.198. Redistribution subject to AECE license or copyright.]

Systems, vol. 24, no. 2, pp. 668–675, May 2009. doi:10.1109/TPWRS.2009.2016457

[3] J.S. Chai, and Anjan Bose. "Bottlenecks in parallel algorithms for power system stability analysis," IEEE Transactions on Power Systems vol. 8 no. 1, pp. 9-15., 1993. doi: 10.1109/59.221242

[4] G. Gurrala, A. Dimitrovski, P. Sreekanth, S. Simunovic, M. Starke, and K. Sun, "Application of adomian decomposition for multi-machine power system simulation," In Power & Energy Society General Meeting, 2015 IEEE, pp. 1-5, 2015., doi: 10.1109/PESGM.2015.7286538

[5] D. Shirmohammadi, H. W. Hong, A. Semlyen, and G. X. Luo, "A compensation-based power flow method for weakly meshed distribution and transmission networks," IEEE Transactions on Power Systems, vol. 3, no. 2, pp. 753–762, May 1988. doi:10.1109/59.192932

[6] D. Capko, A. Erdeljan, M. Popovic, and G. Svenda, "An Optimal Initial Partitioning of Large Data Model in Utility Management Systems," Advances in Electrical and Computer Engineering, vol. 11, no. 4, pp. 41–46, 2011. doi:10.4316/AECE.2011.04007

[7] D. Capko, A. Erdeljan, S. Vukmirovic, and I. Lendak, "A Hybrid Genetic Algorithm for Partitioning of Data Model in Distribution Management Systems," Information Technology And Control, vol. 40, no. 4, Dec. 2011. doi:10.5755/j01.itc.40.4.981

[8] L. Martinovic, D. Capko, and A. Erdeljan, "Estimation methods of calculations complexity in distribution management systems," 2014, pp. 325–328. doi:10.1109/CINTI.2014.7028697

[9] K. Benmohammed-Mahieddine, "An Evaluation of Load Balancing Algorithms for Distributed Systems," PhD thesis, University of Leeds, Oct. 1991.

[10] S. Sharma, S. Singh, M. Sharma, "Performance analysis of load balancing algorithms," In: 38th World Academy of Science, Engineering and Technology, 2008.

[11] S. Zhou, D. Ferrari, "A measurement study of load balancing performance," Proc. 7th Int. Conf. Dist. Computing Syst., pp 490–497, 1987.

[12] R. L. Graham, "Bounds on Multiprocessing Timing Anomalies," SIAM Journal on Applied Mathematics, vol. 17, no. 2, pp. 416–429, Mar. 1969. doi:10.1137/0117039

[13] N. Karmarkar, R. M. Karp, "The differencing method of set partitioning," Technical Report UCB/CSD 82/113, Computer Science Division, University of California, Berkeley, vol. 82, no. 113, pp.181-203, 1982.

[14] R. E. Korf, "Multi-way number partitioning," in Proceedings of the 20nd International Joint Conference on Artificial Intelligence (IJCAI-09), pp. 538–543, 2009.

[15] J. Vaščák, "Adaptation of fuzzy cognitive maps by migration algorithms," Kybernetes, vol. 41, no. 3/4, pp. 429–443, Apr. 2012. doi:10.1108/03684921211229505

[16] R.-E. Precup, M.-C. Sabau, and E. M. Petriu, "Nature-inspired optimal tuning of input membership functions of Takagi-Sugeno-Kang fuzzy models for Anti-lock Braking Systems," Applied Soft Computing, vol. 27, pp. 575–589, Feb. 2015. doi:10.1016/j.asoc.2014.07.004

[17] S. Vrkalovic, T.-A. Teban, and L.-D. Borlea, "Stable Takagi-Sugeno fuzzy control designed by optimization," International Journal of Artificial Intelligence, vol. 15, no. 2, pp. 17–29, 2017.

[18] Z. Chen, S. Zhou, and J. Luo, "A robust ant colony optimization for continuous functions," Expert Systems with Applications, vol. 81, pp. 309–320, Sep. 2017., doi: 10.1016/j.eswa.2017.03.036

[19] J. L. Bosque, P. Toharia, O. D. Robles, and L. Pastor, "A load index and load balancing algorithm for heterogeneous clusters," The Journal of Supercomputing, vol. 65, no. 3, pp. 1104–1113, Sep. 2013. doi:10.1007/s11227-013-0881-3

[20] A. Karimi, F. Zarafshan, A. Jantan, A. R. Ramli, M. Saripan, "A New Fuzzy Approach for Dynamic Load Balancing Algorithm," International Journal of Computer Science and Information Security, IJCSIS, Vol. 6, No. 1, pp. 1-5, Oct. 2009.

[21] P. Kanungo, "Measuring performance of dynamic load balancing algorithms in distributed computing applications," International Journal of Advanced Research in Computer and Communication Engineering, Vol. 2, No. 10, Oct. 2013.

[22] R. Luling, B. Monien, and F. Ramme, "Load balancing in large networks: a comparative study," 1991, pp. 686–689. doi:10.1109/SPDP.1991.218196

[23] E. Laskowski, M. Tudruj, R. Olejnik, and D. Kopanski, "Dynamic Load Balancing Based on Applications Global States Monitoring," 2013, pp. 11–18. doi:10.1109/ISPDC.2013.11

[24] D. Ferrari, S. Zhou, "A load index for dynamic load balancing," in Proceedings of 1986 ACM Fall joint computer conference. IEEE Computer Society Press, pp. 684-690, 1986.

[25] D. Ferrari, S. Zhou, "An empirical investigation of load indices for load balancing applications Proc. Performance," `87, 12th IFIP WG7.3 Int. Symp. on Computer Performance, Brussels, 1987

[26] K. Schloegel, G. Karypis, and V. Kumar, "Multilevel Diffusion Schemes for Repartitioning of Adaptive Meshes," Journal of Parallel and Distributed Computing, vol. 47, no. 2, pp. 109–124, Dec. 1997. doi:10.1006/jpdc.1997.1410

[27] N. Widell, "Migration algorithms for automated load balancing," in Proceedings of 16th International Conf. on Parallel and Distributed Computing and Systems: MIT Cambridge USA, Nov. 2004.

[28] K. Schloegel, G. Karypis, and V. Kumar, "Wavefront diffusion and LMSR: algorithms for dynamic repartitioning of adaptive meshes," IEEE Transactions on Parallel and Distributed Systems, vol. 12, no. 5, pp. 451–466, May 2001. doi:10.1109/71.926167