Scalability Evaluation of NSLP Algorithm for Solving Non-Stationary Linear Programming Problems on Cluster Computing Systems

Irina Sokolinskaya and Leonid B. Sokolinsky✉*

South Ural State University
76 Lenin prospekt, Chelyabinsk, Russia, 454080
`Irina.Sokolinskaya@susu.ru`, `Leonid.Sokolinsky@susu.ru`

Abstract. The paper is devoted to a scalability study of the NSLP algorithm for solving non-stationary high-dimension linear programming problem on the cluster computing systems. The analysis is based on the BSF model of parallel computations. The BSF model is a new parallel computation model designed on the basis of BSP and SPMD models. The brief descriptions of the NSLP algorithm and the BSF model are given. The NSLP algorithm implementation in the form of a BSF program is considered. On the basis of the BSF cost metric, the upper bound of the NSLP algorithm scalability is derived and its parallel efficiency is estimated. NSLP algorithm implementation using BSF skeleton is described. A comparison of scalability estimations obtained analytically and experimentally is provided.

Keywords: non-stationary linear programming problem · large-scale linear programming · NSLP algorithm · BSF parallel computation model · cost metric · scalability bound · parallel efficiency estimation.

1 Introduction

The Big Data phenomenon has spawned the large-scale linear programming (LP) problems [1]. Such problems arise in the following areas: scheduling, logistics, advertising, retail, e-commerce [2], quantum physics [3], asset-liability management [4], algorithmic trading [5,6,7,8] and others. The similar LP problems include up to tens of millions of constraints and up to hundreds of millions of decision variables. In many cases, especially in mathematical economy, these LP problems are nonstationary (dynamic). It means that input data (matrix A, vectors b and c) is evolving with time, and the period of data change is within the range of hundredths of a second.

---

2   I. Sokolinskaya, L.B. Sokolinsky

Until now, one of the most popular methods solving LP problems is the class of algorithms proposed and designed by Dantzig on the base of the simplex method [9]. The simplex method has proved to be effective in solving a large class of LP problems. However, in certain cases the simplex method has to move across all the vertices of the polytope, which corresponds to an exponential time complexity [10]. Karmarkar in [11] proposed a method for linear programming called "Interior point method" which runs in polynomial time and is also very efficient in practice.

The simplex method and the method of interior points remain today the main methods for solving the LP problem. However, these methods may prove ineffective in the case of large scale LP problems with rapidly evolving input data. To overcome the problem of non-stationarity of input data, the authors proposed in [12] the scalable algorithm *NSLP* (*Non-Stationary Linear Programming*) for solving large-scale non-stationary LP problems on cluster computing systems. It includes two phases: *Quest* and *Targeting*. The *Quest* phase calculates a solution of the system of inequalities defining the constraint system of the linear programming problem under condition of the dynamic changes of input data. The point of pseudo-projection on n-polytope M is taken as a solution. Polytope M is the set of feasible solutions of the LP problem. The pseudo-projection is an extension of the projection, which uses Fejer (relaxation) iterative process [13,14,15,16]. A distinctive feature of the Fejer process is its "self-guided" capability: the Fejer process automatically corrects its motion path according to the polytope position changes during the calculation of the pseudo-projection. The *Quest* phase was investigated in [12], where the convergence theorem was proved for the case when the polytope is translated with a fixed vector in the each unit of time. In the paper [17], the authors demonstrated that Intel Xeon Phi multi-core processors can be efficiently used for calculating the pseudo-projections.

The *Targeting* phase forms a special system of points having the shape of the n-dimensional axisymmetric cross. The cross moves in the n-dimensional space in such a way that the solution of the LP problem permanently was in the ε-vicinity of the central point of the cross. The Targeting phase can be effectively implemented as a parallel program for a clustered computing system by using the "master-workers" framework [18,19,20]. In this paper, we discuss a parallel implementation of the NSLP algorithm using the BSF computational model presented in [21]. On the base of the described BSF-implementation, a quantitative scalability analysis of the NSLP algorithm is performed.

The rest of the paper is organized as follows. Section 2 gives a formal statement of a LP problem and presents the brief description of the NSLP algorithm. Section 3 provides an outline of the BSF computational model and presents corresponding cost metrics. Section 4 describes a BSF-implementation of the NSLP algorithm, calculates the upper bound of scalability and evaluates the parallel efficiency depending on the percentage of initial data being changed dynamically. Section 5 describes an implementation of the NSLP algorithm based on the BSF skeleton in C language and compares the results obtained analytically

and experimentally. Section 6 summarizes the results obtained and proposes the directions for future research.

2   NSLP algorithm

Let we be given a non-stationary LP problem in the vector space $\mathbb{R}^n$:

$$\max\left\{\langle c_t, x\rangle \,|\, A_t x \le b_t, \ x \ge 0\right\}, \tag{1}$$

where the matrix A_t has m rows. The non-stationarity of the problem means that the values of the elements of the matrix A_t and the vectors b_t, c_t depend on the time $t \in \mathbb{R}_{\ge 0}$. We assume that the value of $t = 0$ corresponds to the initial instant of time:

$$A_0 = A, b_0 = b, c_0 = c \tag{2}$$

Let M_t be a polytope defined by the constraints of the non-stationary LP problem (1). Such a polytope is always convex. The *Quest* phase calculates a point z belonging to the polytope M_t. This phase is described in detail in [12]. The *Quest* Phase is followed by the *Targeting* phase. At the *Targeting* phase, a n-dimensional axisymmetric cross is formed. The *n-dimensional axisymmetric cross* is a finite set $G = \{g_0, \ldots, g_{P-1}\} \subset \mathbb{R}^n$ having the cardinality equals $P+1$, where P is a multiple of $n \ge 2$. Among points of the cross, the point g_0 called the *central point* is single out. The initial coordinates of the central point are assigned the coordinates of the point z calculated in the *Quest* phase. The set $G\backslash\{g_0\}$ is divided into n disjoint subsets C_i ($i = 0, \ldots, n-1$) called the *cohorts*:

$$G\backslash\{g_0\} = \bigcup_{i=0}^{n-1} C_i.$$

Each i-th cohort ($i = 0, \ldots, n-1$) consists of

$$K = P/n \tag{3}$$

points lying on the straight line, which is parallel to the i-th coordinate axis and passing through the central point g_0. By itself, the central point does not belong to any cohort. The distance between any two neighbor points of the set $G \cup \{g_0\}$ is equal to the constant s. It can be changed during computing. An example of the two-dimensional cross is shown in Fig. 1. The number of points in one dimension excluding the central point is equal to K. The symmetry of the cross supposes that K takes only even values greater than or equal to 2. Using equation (3), we obtain the following equation giving the total number of points in the cross:

$$P + 1 = nK + 1 \tag{4}$$

Since K can take only even values greater than or equal to 2 and $n \ge 2$, from equation (4), it follows that P can also take only even values and $P \ge 4$. In Fig. 1, we have $n = 2$, $K = 6$, $P = 12$.

4   I. Sokolinskaya, L.B. Sokolinsky
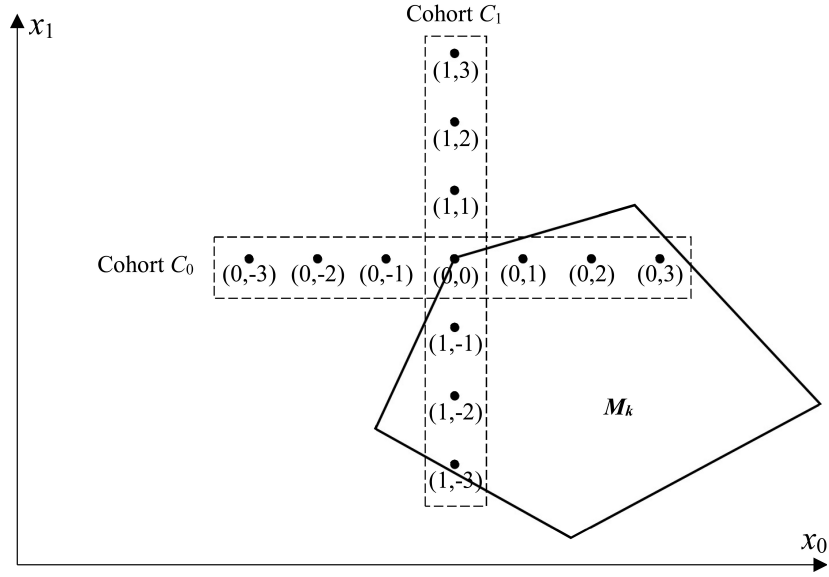
Fig. 1. Two-dimensional cross G: $K = 6$, $P = 12$

Each point of the cross G is uniquely identified by a *marker* being a pair of integers numbers (χ, η) such that $0 \leq \chi < n$, $|\eta| \leq K/2$. Informally, χ specifies the number of the cohort, and η specifies the sequence number of the point in the cohort C_χ, being counted out of the central point. The corresponding marking of points for the two-dimensional case is given in Fig. 1. The coordinates of the point $x_{(\chi, \eta)}$ having the marker (χ, η) can be reconstructed as follows:

$$x_{(\chi, \eta)} = g_0 + (0, \ldots, 0, \underbrace{\eta \cdot s}_{\chi}, 0, \ldots, 0) \tag{5}$$

The vector being added to g_0 in the right part of the equation (5) has a single non-zero coordinate in the position χ. This coordinate equals $\eta \cdot s$, where s is the distance between neighbor points in a cohort.

The *Targeting* phase includes the following steps.

1. Build the n-dimensional axisymmetric cross G that has K points in each cohort, the distance between neighbor points equaling s, and the center at point $g_0 = z_k$, where z_k is obtained in the *Quest* phase.
2. Calculate $G' = G \cap M_k$.
3. Calculate $C'_\chi = C_\chi \cap G'$ for $\chi = 0, \ldots, n - 1$.
4. Calculate $Q = \bigcup\limits_{\chi=0}^{n-1} \{\arg\max \{\langle c_k, g \rangle \mid g \in C'_\chi, C'_\chi \neq \emptyset\}\}$.
5. If $g_0 \in M_k$ and $\langle c_k, g_0 \rangle \geq \max\limits_{q \in Q} \langle c_k, q \rangle$, then $k := k + 1$, and go to step 2.
6. $g_0 := \dfrac{\sum\limits_{q \in Q} q}{|Q|}$.

7. $k := k + 1$.
8. Go to step 2.

Thus, in the Targeting phase, the steps 2–7 form a perpetual loop in which the approximate solution of the non-stationary LP problem is permanently recalculated. From the non-formal point of view, in the step 2, we determine which points of the cross G are belonged to the polytope M_k. In the step 3, points that do not belong to the polytope are dropped out of each cohort. In the step 4, the point with the maximum value of the objective function is chosen among the residuary points of each cohort. In the step 5, we check if the value of the objective function at the central point of the cross is greater than all the maximums found in the step 4. If this condition is true then the cross does not shift, the time counter t is incremented by one unit and the next iteration is started. If this condition is false then we go to step 6 where the new center point is calculated as the centroid of the set of points obtained in the step 4. In the step 7, the time counter t is incremented by one unit. In the step 8, we go to the new iteration. In such a way, the center g_0 of the cross G permanently performs the role of an approximate solution of the non-stationary problem (1).

3   BSF computational model

We use the BSF parallel computation model proposed in [21] to evaluate the upper bound of the scalability of the NSLP algorithm in the Targeting phase. The BSF (Bulk Synchronous Farm) model was proposed to multiprocessor systems with distributed memory. A *BSF-computer* consists of a collection of homogeneous computing nodes with private memory connected by a communication network that delivers messages among the nodes. Among all the computing nodes, one node called the *master-node* is single out. The rest of the nodes are the *slave-nodes*. The BSF-computer must include at least one master-node and one slave-node. Thus, if P is the number of slave-nodes then $P \geq 1$.

BSF-computer utilizes the *SPMD* programming model [22] according to which all nodes executes the same program but process different data. A BSF-program consists of sequences of macro-steps and global barrier synchronizations performed by the master and all the slaves. Each macro-step is divided into two sections: master section and slave section. A master section includes instructions performed by the master only. A slave section includes instructions performed by the slaves only. The sequential order of the master section and the slave section within the macro-step is not important. All the slave nodes act on the same data array, but the base address of the data assigned to the slave-node for processing is determined by the logical number of this node. The BSF-program includes the following sequential sections (see Fig. 2):

– initialization;
– iterative process;
– finalization.
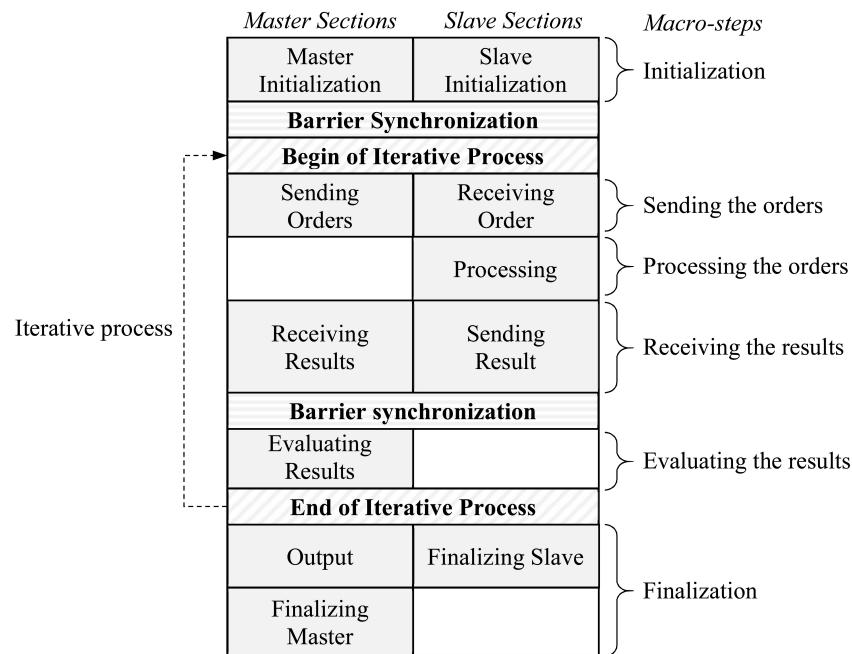
6   I. Sokolinskaya, L.B. Sokolinsky

Master Sections	Slave Sections	Macro-steps
Master Initialization	Slave Initialization	Initialization
Barrier Synchronization		
Begin of Iterative Process		
Sending Orders	Receiving Order	Sending the orders
	Processing	Processing the orders
Receiving Results	Sending Result	Receiving the results
Barrier synchronization		
Evaluating Results		Evaluating the results
End of Iterative Process		
Output	Finalizing Slave	Finalization
Finalizing Master		

Iterative process

Fig. 2. BSF-program structure

Initialization is a macro-step, during which the master and slaves read or generate input data. The initialization is followed by a barrier synchronization. The *iterative process* repeatedly performs its body until the exit condition checked by the master becomes true. In the *finalization macro-step*, the master outputs the results and ends the program.

Body of the iterative process includes the following macro-steps:

1) sending the orders (from master to slaves);
2) processing the orders (slaves);
3) receiving the results (from slaves to master);
4) evaluating the results (master).

In the first macro-step, the master sends the same orders to all the slaves. Then, the slaves execute the received orders (the master is idle at that time). All the slaves execute the same program code but act on the different data with the base address depending on the slave-node number.

It means that all slaves spend the same time for calculating. During processing the order, there are no data transfers between nodes. In the third step, all slaves send the results to the master. After that, global barrier synchronization is performed. During the fourth step, the master evaluates received results. The slaves are idle at that time. After result evaluations, the master checks the exit condition. If the exit condition is true then iterative process is finished, otherwise the iterative process is continued.

The BSF model provides an analytical estimation of the scalability of a BSF-program. The main parameters of the model are [21]:

P: the number of slave-nodes;

L: an upper bound on the latency, or delay, incurred in communicating a message containing one byte from its source node to its target node;

t_s: time that the master-node is engaged in sending one order to one slave-node excluding the latency;

t_v: time that a slave-node is engaged in execution an order within one iteration (BSF-model assumes that this time is the same for all the slave-nodes and it is a constant within the iterative process;

t_r: total time that the master-node is engaged in receiving the results from all the slave-nodes excluding the latency;

t_p: total time that the master-node is engaged in evaluating the results received from all the slave-nodes.

Lets denote $t_w = P \cdot t_v$ – summarized time which is spent by slave-nodes for order executions. Then, the upper bound of a BSF-program scalability can be estimated by the following inequality [21]:

$$P \le \sqrt{\frac{t_w}{2L + t_s}}. \tag{6}$$

Note that the upper bound of the BSF-program scalability does not depend on the time, which the master is engaged in receiving and evaluating the slave results. The speedup of BSF-program can be calculated by the following equation [21]:

$$a = \frac{P(2L + t_s + t_r + t_p + t_w)}{P^2(2L + t_s) + P(t_r + t_p) + t_w}. \tag{7}$$

One more important property of a parallel program is the parallel efficiency. The parallel efficiency of a BSF-program can be calculated by the following approximate equation [21]:

$$e \approx \frac{1}{1 + (P^2(2L + t_s) + P(t_r + t_p))/t_w}. \tag{8}$$

4   BSF-implementation of NSLP algorithm

In this section, we demonstrate how the algorithm presented in section 2 can be implemented in the BSF-program form. Based on this implementation, we calculate the time complexity of one iteration and give an analytical estimation of the scalability upper bound of the NSLP algorithm in the Targeting phase. For calculating, we use the synthetic scalable linear programming problem of the dimension called *Model-n* [17]. This LP problem has the matrix A of the size $n \times 2(n + 1)$. We assume $n > 10^4$.

The NSLP algorithm can be implemented in the BSF-program form by the following way. In the Initialization macro-step, the master and all the slaves

read (generate) and store in the local memory all the initial data of the non-stationary LP problem (1); the master executes the Quest phase and finds a point z belonging to the polytope M_t. Then, the iterative process begins. In each iteration, the following steps are performed:

1) sending the orders from master to slaves;
2) processing the orders by slaves;
3) sending the results from slaves to master;
4) evaluating the results by master.

Table 1. Structure of message "Order for slaves"

No.	Attribute ID	Attribute semantic	Overhead
1	θ	New central point coordinates of the n-dimensional cross	t_θ
2	α	New values of matrix A entries	t_α
3	β	New values of column b elements	t_β
4	γ	New values of objective function co-efficients	t_γ

The *order* includes the information given in Table 1. Suppose that the fraction of the changed elements of matrix A, column b and objective function c coefficients equals to $\delta(n)$, where $\forall n \, (0 \le \delta(n) \le 1)$. In that case, the time t_s that the master-node is engaged in sending one order to one slave-node (excluding the latency) can be approximated according to Table 1 as follows:

$$
\begin{aligned}
t_s = t_\theta + t_\alpha + t_\beta + t_\gamma &= O(n) + O(\delta(n) \cdot n \cdot 2(n+1)) + O(\delta(n) \cdot 2(n+1)) + O(\delta(n)n) = \\
&= O(n) + O(\delta(n) \cdot n(n+1)) + O(\delta(n)(n+1)) + O(\delta(n)n) < \\
&< O(n+1) + O(\delta(n) \cdot (n+1)^2) + O(\delta(n)(n+1)) + O(\delta(n)(n+1)) = \\
&= O(n+1) + O(\delta(n) \cdot (n+1)^2) + O(\delta(n)(n+1)) < \\
&< O(\delta(n) \cdot (n+1)^2) + O(n+1).
\end{aligned}
$$

Hence,
$$
t_s < O(\delta(n) \cdot (n+1)^2) + O(n+1). \tag{9}
$$

The smallest unit of parallelization in the BSF-implementation of the NSLP algorithm is the cohort. The number of cohorts equals to the space dimension n. Thus, the number P of slave-nodes should be less than or equal to the space dimension n. We shall assume $n \gg P$. A slave-node sequentially process all the cohorts assigned to it. In the current cohort, the coordinates of every point x are calculated using equation (5). The time complexity of this operation is $O(n)$. Then the point x is checked to be belonged to the polytope M_t. To do this, it

is sufficient for the slave to verify the truth of the condition $A_t x = b_t$. Since A_t is of size $n \times 2(n + 1)$, the time complexity of this operation is $O(n^2 + n)$. The number of points in a cohort excluding the central one is equal to the constant K. According to the equation (4), the total number of points in the cross excluding the central one is equal to nK. Hence, the time complexity of the calculations performed for all points of the cross in steps 2–3 can be estimated as $O(n^3 + n^2)$. After this, the slaves partially (for their cohorts only) execute the step 4 of Targeting phase (see Section 2). The total time complexity of these operations is $O(n^2)$. Thus, the total time complexity of all the calculations performed by slaves has the following estimation:

$$t_w = O(n^3 + n^2) + O(n^2) + O(n) \leq O(n^3 + n^2 + n). \tag{10}$$

As a result, each slave sends to the master a summarized vector of points which belong to the polytope and have the maximum value of the objective function in the corresponding cohort. Thus, the total time complexity of transferring the results from the slaves to the master is

$$t_r = O(PKn) \approx O(n). \tag{11}$$

Having received the results from the slaves, the master sums them up to complete the step 4 of the algorithm. The time complexity of these calculations will have the following estimation:

$$t_{\text{step 4}} \approx O(n^2). \tag{12}$$

Because of the non-stationarity of the LP problem, the condition in step 5 will be rarely true. Hence, we may assume that the next step after the step 4 will be the step 6 in most cases. Since the number of cohorts equals to n, the total time complexity of the step 6 of the Targeting phase is

$$t_{\text{step 6}} = O(n^2). \tag{13}$$

Thus, the total time complexity of processing the results obtained by the master from the slaves is

$$t_p \approx t_{\text{step 4}} + t_{\text{step 6}} = O(n^2). \tag{14}$$

Substituting the values from (10) and (9) into equation (6), we obtain the following estimation for the upper bound of the NSLP algorithm scalability:

$$P_{NSLP} \leq \sqrt{\frac{O(n^3 + n^2 + n)}{2L + O(\delta(n) \cdot (n + 1)^2) + O(n + 1)}}. \tag{15}$$

Suppose all the input data of the problem are changed at each iteration. It corresponds to $\delta(n) = 1$. In this case, inequality (15) is converted to the following form

$$P_{NSLP} \leq \sqrt{\frac{O(n^3 + n^2 + n)}{2L + O((n + 1)^2) + O(n + 1)}} \approx O(\sqrt{n}). \tag{16}$$

10   I. Sokolinskaya, L.B. Sokolinsky

It means that the upper bound of the BSF-program scalability increases proportionally to the square root of the problem dimension. Hence, the NSLP algorithm implementation in the form of a BSF-program has limited scalability in this case.

Now suppose that the fraction of the changed problem input data at each iteration is

$$\delta(n) = \frac{1}{2(n+1)}. \tag{17}$$

It corresponds to a situation where the matrix has only one changed row, the column has only one changed element, and the objective function has no more than one changed coefficient. In this case, we obtain the following estimation

$$P_{NSLP} \leq \sqrt{\frac{O(n^3 + n^2 + n)}{2L + O(n+1) + O(n+1)}} \approx O(\sqrt{n^2}) = O(n) \tag{18}$$

substituting the value of $\delta(n)$ from the equation (17) into the equation (15). It means that the upper bound of the BSF-program scalability increases proportionally to the problem dimension. Hence, the NSLP algorithm implementation in the form of a BSF-program is scalable well in this case.

We can also estimate the BSF-implementation parallel efficiency of the NSLP algorithm using approximate equation (8):

$$e = \frac{1}{1 + \frac{P^2 \cdot (2L + t_s) + P \cdot (t_r + t_p)}{t_w}} =$$

$$= \frac{1}{1 + \frac{P^2 \cdot (2L + O(\delta(n) \cdot (n+1)^2) + O(n+1)) + P \cdot (O(n) + O(n^2))}{O(n^3 + n^2)}}. \tag{19}$$

For $\delta(n) = 1$, $n \to \infty$ and $P \to \infty$, we get from (19) the following estimation

$$e = \frac{1}{1 + \frac{P^2(2L + O((n+1)^2) + O(n+1)) + P(O(n) + O(n^2))}{O(n^3 + n^2)}} \approx$$

$$\approx \frac{1}{1 + \frac{P^2(O(n^2) + O(n)) + P \cdot (O(n^2) + O(n))}{O(n^3 + n^2)}} = \tag{20}$$

$$= \frac{1}{1 + (P^2 + P)\frac{O(n^2) + O(n)}{O(n^3 + n^2)}} \approx \frac{1}{1 + \frac{P^2 + P}{O(n)}} \approx \frac{1}{1 + P^2/O(n)}.$$

In such a way, we obtain

$$e \approx \frac{1}{1 + P^2/O(n)}. \tag{21}$$

Hence for $\delta(n) = 1$, the high parallel efficiency is achieved when $n \gg P^2$.

For $\delta(n) = \frac{1}{2(n+1)}$ we get from (19) the following estimation

$$e \approx \frac{1}{1 + P^2/O(n^2) + P/O(n)}. \tag{22}$$

Hence for $\delta(n) = \frac{1}{2(n+1)}$, the high parallel efficiency is achieved when $n \gg P$.

(a) Speedup

(b) Parallel efficiency
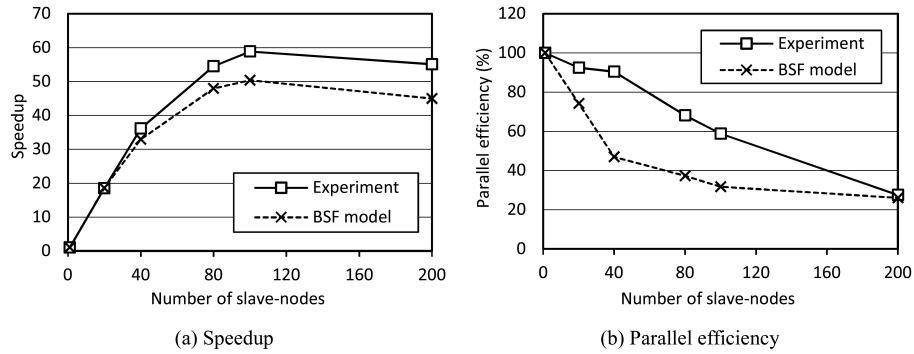
Fig. 3. Experiments for $n = 400$.

(a) Speedup

(b) Parallel efficiency
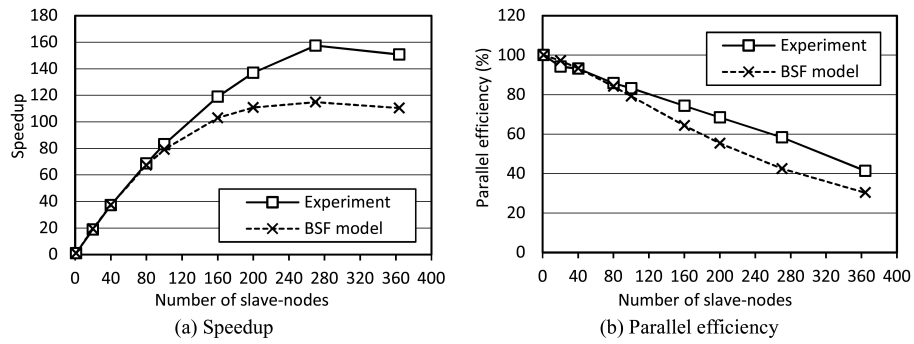
Fig. 4. Experiments for $n = 800$.

(a) Speedup

(b) Parallel efficiency

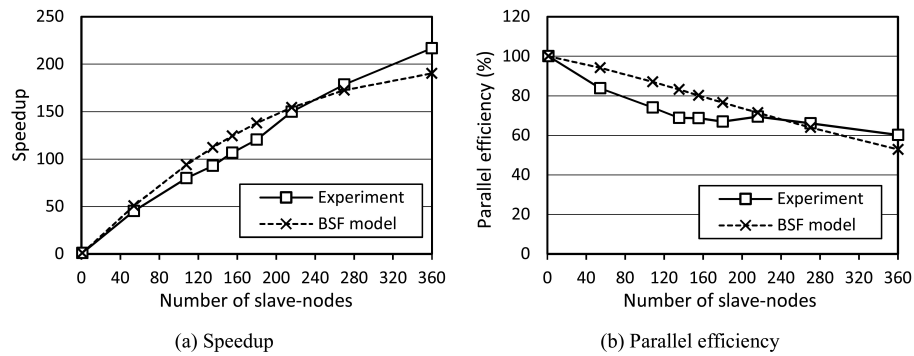**Fig. 5.** Experiments for $n = 1080$.

5   Numerical Experiments

The implementation of the Qwest phase was described and evaluated by us in the paper [17]. In the present work, we have done the implementation of the Targeting phase in C language using the BSF skeleton. The source code of this program is freely available on Github, at https://github.com/leonid-sokolinsky/ BSF-NSLP. We investigated the speedup and parallel efficiency of this BSF-program on the supercomputer "Tornado SUSU" [23] using the synthetic scalable linear programming problem *Model n* [17] mentioned in the section 4. The calculations were performed for the dimensions 400, 800 and 1080. At the same time, we plotted the curves of speedup and parallel efficiency for these dimensions using equations (7) and (8). We assumed that $\delta(n) = 1/2(n + 1)$. The results are presented in Fig. 3–5. In all cases, the analytical estimations were very close to experimental ones. Moreover, the performed experiments show that the upper bound of the BSF-program scalability increases proportionally to the problem dimension. It was analytically predicted using the equation (18) in Section 4.

6   Conclusion

In this paper, the scalability and parallel efficiency of the NSLP algorithm used to solve large-scale non-stationary linear programming problems on cluster computing systems were investigated. To do this, we used the BSF (Bulk Synchronous Farm) parallel computation model based on the "master-slave" paradigm. The BSF-implementation of the NSLP algorithm is described. A scalability upper bound of the BSF-implementation of the NSLP algorithm is obtained. This estimation tells us the following. If all the input data of the problem are changed at each iteration then the upper bound of the BSF-program scalability increases proportionally to the square root of the problem dimension. In this case, the NSLP algorithm implementation in the form of a BSF-program has limited scalability. If each inequality of the constraint system has no more than one coefficient changed during an iteration then the upper bound of the BSF-program scalability increases proportionally to the problem dimension. In this case, the NSLP algorithm implementation in the form of a BSF-program is scalable well. The equations for estimating the parallel efficiency of the BSF-implementation of the NSLP algorithm are also deduced. These equations allow us to conclude the following. If during the iteration all the problem input data are dynamically changed then for the high parallel efficiency it is necessary that the problem dimension is much greater than the square of the number of slaves: $n \gg P^2$. However, if in each inequality of the constraint system no more than one coefficient changes during each iteration then for a high parallel efficiency it is necessary that the problem dimension be much greater than the number of slaves: $n \gg P$. The numerical experiments with a synthetic problem showed that the BSF model accurately predicts the upper bound of the scalability of the program that implements the Targeting phase using the BSF skeleton.

As future research directions, we intend to do the following:

1) implement the Qwest phase in C language using the BSF skeleton and MPI-library;
2) carry out numerical experiments on a cluster computing system using synthetic and real LP problems;
3) compare the scalability boundaries of the Qwest phase obtained experimentally and analytically.

References

1. Chung, W.: Applying large-scale linear programming in business analytics. In: Proceedings of the 2015 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM), pp. 1860-1864. IEEE (2015)
2. Tipi, H.: Solving super-size problems with optimization. Presentation at the meeting of the 2010 INFORMS Conference on O.R. Practice. Orlando, Florida. April 2010. http://nymetro.chapter.informs.org/prac_cor_pubs/06-10%20Horia%20Tipi%20SolvingLargeScaleXpress.pdf (accessed 07.05.2017).
3. Gondzio, J. et al.: Solving large-scale optimization problems related to Bells Theorem. Journal of Computational and Applied Mathematics, vol. 263, pp. 392-404. (2014)
4. Sodhi, M.S.: LP modeling for asset-liability management: A survey of choices and simplifications. Operations Research, vol. 53, no. 2, pp. 181-196. (2005)
5. Brogaard, J., Hendershott, T., Riordan, R.: High-Frequency Trading and Price Discovery. Review of Financial Studies. 27, 2267–2306 (2014).
6. Budish, E., Cramton, P., Shim, J.: The High-Frequency Trading Arms Race: Frequent Batch Auctions as a Market Design Response. The Quarterly Journal of Economics. 130, 1547–1621 (2015).
7. Gomber, P., Arndt, B., Lutat, M., Uhle, T.E.: High-Frequency Trading. SSRN Electronic Journal. 86 (2011).
8. Hendershott, T., Jones, C.M., Menkveld, A.J.: Does Algorithmic Trading Improve Liquidity? The Journal of Finance. 66, 1–33 (2011).
9. Dantzig, G.B.: Linear programming and extensions. Princeton university press, Princeton, N.J. (1998).
10. Klee, V., Minty, G.J.: How good is the simplex algorithm? In: Inequalities III (Proceedings of the Third Symposium on Inequalities held at the University of California, Los Angeles, Calif., September 1–9, 1969, dedicated to the memory of Theodore S. Motzkin). pp. 159–175. Academic Press, New York-London (1972).
11. Karmarkar, N.: A new polynomial-time algorithm for linear programming. Combinatorica. 4, 373–395 (1984).
12. Sokolinskaya, I., Sokolinsky, L.B.: On the Solution of Linear Programming Problems in the Age of Big Data, http://arxiv.org/abs/1706.10030.
13. Agmon, S.: The relaxation method for linear inequalities. Canadian Journal of Mathematics. 6, 382–392 (1954).
14. Motzkin, T.S., Schoenberg, I.J.: The relaxation method for linear inequalities. Journal canadien de mathmatiques. 6, 393–404 (1954).
15. Eremin, I.I.: Fejerovskie metody dlya zadach linejnoj i vypukloj optimizatsii [Fejers Methods for Problems of Convex and Linear Optimization]. Publishing of the South Ural State University, Chelyabinsk (2009).
16. Gonzlez-Gutirrez, E., Hernndez Rebollar, L., Todorov, M.I.: Relaxation methods for solving linear inequality systems: converging results. TOP. 20, 426–436 (2012).

14   I. Sokolinskaya, L.B. Sokolinsky

17. Sokolinskaya, I., Sokolinsky, L.: Revised Pursuit Algorithm for Solving Non-stationary Linear Programming Problems on Modern Computing Clusters with Manycore Accelerators. In: Supercomputing. RuSCDays 2016. Communications in Computer and Information Science. pp. 212–223. Springer International Publishing, Cham (2016).
18. Sahni, S., Vairaktarakis, G.: The master-slave paradigm in parallel computer and industrial settings. Journal of Global Optimization. 9, 357–377 (1996).
19. Silva, L.M., Buyya, R.: Parallel programming models and paradigms. In: High Performance Cluster Computing: Architectures and Systems. Vol. 2. pp. 4–27 (1999).
20. Leung, J.Y.-T., Zhao, H.: Scheduling problems in master-slave model. Annals of Operations Research. 159, 215–231 (2008).
21. Sokolinsky, L.B.: Analytical study of the master-worker framework scalability on multiprocessors with distributed memory, http://arxiv.org/abs/1704.05816.
22. Darema, F., George, D.A., Norton, V.A., Pfister, G.F.: A single-program-multiple-data computational model for EPEX/FORTRAN. Parallel Computing. 7, 11–24 (1988).
23. Kostenetskiy, P.S., Safonov, A.Y.: SUSU Supercomputer Resources. In: Proceedings of the 10th Annual International Scientific Conference on Parallel Computing Technologies (PCT 2016). CEUR Workshop Proceedings. Vol. 1576. pp. 561–573 (2016).