Distributed computing systems synchronization modeling for solving machine learning tasks

T V Azarnova and P V Polukhin
Department of Applied Mathematics and Informatics and Mechanics, Voronezh State University, Voronezh, Russia

E-mail: ivdas92@mail.ru

Abstract. Distributed computing systems are an effective tool for solving complex problems related to processing large amounts of information and data, in particular, machine learning tasks. To improve the efficiency of these systems in solving complex tasks, special methods for synchronization processes modeling, evaluating and predicting processing delays and query execution, developed in the terms of this models. The approaches for the distributed computing systems organization based on the mathematical apparatus of queuing theory, considered in this paper, allow optimizing the requests processing mechanisms related to solving resource allocation problems, and increasing the consistency of computational processes related to machine learning problems.

1. Introduction
The learning problems solution implemented within the context of setting parameters of artificial neural networks, Bayesian networks and decision-making networks requires significant computational resources, when implementing different tasks such as: generation of the initial probability distribution (dynamic Bayesian networks), learning and distribution of probabilistic inference. In the process of the various algorithms operation oriented to obtain the probability distribution with the required accuracy, especially based on the Monte Carlo method, it is necessary to form a sufficiently large number of samples. The scale of calculations creates difficulties in carrying out practical calculations. Modern computers have significant resource capabilities, but within the framework of generating and learning large amounts of data, their resources often become insufficient for the complex operational performance tasks. Distributed systems with a common bus are used to solve these problems. Standard data transmission systems based on Ethernet and Fiber Channel lines (FC) can act as a bus. A special role in the design of distributed systems is occupied by solving issues of data synchronization, load balancing, preventing collisions and errors that arise during the distributed systems activity. Synchronization processes performed within parallel algorithms, generate streams of requirements, allocate computing resources, as well as gaining access to a data area, in particular, learning samples loaded from the file system and distributed among individual parallel blocks of the algorithm. Modeling the parallel algorithms executing process based on queuing theory (QT) allows to increase the efficiency of their mutual influence, optimize delays, and reduce the level of processed data inconsistency in distributed systems. The application of queuing theory allows to calculate the main efficiency characteristics of the considered parallel computing systems, to determine the main factors of stability when the failure or error events come up.
2. Distributed systems synchronization algorithms

Among the synchronization algorithms of distributed systems, algorithms based on sending broadcast messages between individual nodes evolve these systems in order to achieve the required level of consistency are particular favour. The most widely used algorithms are Lamport, Rickart-Agrawal and Suzuki-Kasami. The first two algorithms use timestamps to synchronize the processing of requests between individual nodes, which necessitate introduce additional logical clocks and assigne a timestamp for each node. This approach requires continual synchronization of timestamps, otherwise the calculation procedure will be impossible. The Suzuki-Kasami algorithm [1] doesn’t use timestamps, it uses markers to determine exclusive requirements access that arise during synchronization. The synchronization criterion is determined by organizing procedure for the token ownership process. The order of receiving token is formed according to the requests received from the other processes within broadcast messages sending procedure, reflecting the demand of processes to access shared resources. The token-passing algorithm that determines the access necessity to resources by some process \( \theta_i \) is performed through sending a privileged message \( P_{R_i} \). At the very beginning, we assume that the ownership of the token belongs to only one process \( \theta_1 \). To acquire the token, any of the processes \( \theta_j \) sends messages \( R_j \) to all the remaining processes \( \theta_i \). At the same time, we introduce a restriction related to the fact that the process \( \theta_j \) has access to the token at the current time and can periodically access it, until a message \( P_{R_i} \) is sent to another process \( \theta_{j+1} \). Let us define the structure of transmitted messages as \( R_j = R(j,n), j=1,2,...,m \). In this case, \( j \) – is a sequence number of the processes for that the message has been sent, \( n \) – a set of numbers that determine the total quantity of calls for the process \( \theta_j \) to implement the entry procedure for the critical section \( n+1 \). From the above statements follows that each process \( n+1 \) for which synchronization messages are sent, it is necessary to store sets that include the maximum values from the \( n \) tuples for the remaining processes \( \theta_{j+m} \). Formalizing the presentation of the update procedure \( RN \) at the time of messages receipt \( R_j = R(j,n) \)

\[
RN[j] = \max(RN[j], n)
\]

To obtain a synchronization model for the Suzuki-Kasami algorithm, consider the order in which messages are generated and sent between processes \( \theta_j \) and \( \theta_{j+m} \). To transmit messages, the format \( P_{R_1}(Q, LN) \) is used, \( Q \) – a queue of requests with a dimension \( N_Q \) for each processes \( \theta_{j+m} \) over that broadcasting is performed, \( LN \) – a queue of requests containing tuples of values, and \( LN[j] \) assign the sequence number of messages specific to the current process \( \theta_i \). As soon as the process \( \theta_i \) exits the critical section and releases access to the distributed resource, the queue is updated \( LN[i] = RN[i] \) for the last received message \( P_{R_i} \), corresponding to the process \( \theta_i \). After that, update the queue \( RN[i] = LN[i] + 1 \) for \( \theta_i \). If \( \theta_i \notin Q \), add \( \theta_i \notin Q \) to the queue \( \theta_i \notin Q \). Later on forming the queue \( RN[i] \)

, if \( Q \neq \emptyset \), send a selective message \( P_{R_1}(Q[N_Q-1], LN) \) to the first of the processes \( \theta_j \in Q \). If the queue satisfy condition \( Q = \emptyset \), the process \( \theta_i \) will hold the token until there is no thread \( \theta_j \) getting from \( Q \) in the process of sending messages \( R_j \) by the process \( \theta_i \). In this case, the queue \( Q \) will contain only those processes from which broadcast messages were received to access shared resources. Meanwhile the set \( RN[j] \) will accumulate massages specific for each process \( \theta_j \in Q \) in accept order. If mutual lock is released, algorithm send broadcast message \( R_j \) that indicate that resource is ready for
processing by another process $\theta_i \in Q$. This procedure is repeated until each process elaborate distributed resource and the set $RN[i]$ has become empty for each process $\theta_j \in Q$. The desired synchronization model based on the Suzuki-Kasami algorithm has the following two-stage representative form

In Figure 1, it is assumed that the processes $\theta_{j,m}$ directly related to the processes $\theta_i$ and $\theta_j$ are marked as a bold line. This makes it possible to determine the direction of the synchronization procedure when we passing token between multiple threads. The complexity of presenting the classical model based on the Suzuki-Kasami algorithm lies in the lack of forecasting and estimation algorithms throughput, as well as ambiguity in determining the maximum dimension of the total requests count that occurs during synchronization. This is associated with fact that parallel systems can run several different jobs distributed among all threads of the system, and the number of inputs that can enter critical sections to ensure synchronization of distributed resources can be unlimited [2]. To solve such problems, it is advisable to use elements of the queuing theory (QT). In this case, the synchronization model will maintain and evaluate the maximum allowable flow of requirements necessary for the normal synchronization algorithm operation. Taking into account the fact that most of the classical synchronization mechanisms can be represented as single-line systems, the research will consider single-line queuing systems (SLQS) with delays and a stationary requirement flow without aftereffects. To specify the QS, we need to define three main parameters: the input flow, the queue discipline, and the request service mechanism. As a queue discipline, we will consider FIFO (the first thread comes is served). Consider the procedure for setting an input stream of requirements. It's supposed that $t= t_1, t_2, \ldots, t_n, \ldots (0 < t_1 < t_2 < \ldots < t_n)$ – the synchronization moments of receiving requests, form a Poisson flow and the duration of request processing subordinate to the exponential law with intensity $\mu$.

If resource synchronize request is received at a time when no one of the processes has been locked this resource, then synchronization is performed for the first thread that accessed the resource, the remaining requests accepted from other threads are queued and will be executed in the receipt order as soon as the first thread releases the resource. The end of the service request associated with a completion of the resource mutual lock by a separate thread and the exit from the critical section. Consider the structure of the incoming requirements flow as a generic set of demands for synchronizing resources between
processes in a distributed computing system. To determine the probability function for the arrival of \( k \) requirements over time \( t - p_k(t) \), we suppose [3]

\[
p_k(t + h) = \sum_{i=0}^{k} p_i(t) p_{k-i}(h), h > 0, t > 0
\]  

(1)

The synchronization process uses a Poisson flow of requirements without aftereffects \( p_0(t) = e^{-\lambda t} \). Therefore, expression (1) is transformed as follows:

\[
p_k(h) = 1 - \lambda h + o(h), h \to 0
\]  

(2)

\[
p_k(t + h) = (1 - \lambda h) p_k(t) + \sum_{i=0}^{k-1} p_i(t) p_{k-i}(h) + o(h)
\]  

(3)

From expression (3), with provision that \( p_{k-n}(h)/h = \lambda a_{k-n} \), the following differential equation can be obtained

\[
\frac{p_k(t + h) - p_k(t)}{h} = -\lambda p_k(t) + \sum_{i=0}^{k-1} p_i(t) \frac{p_{k-i}(h)}{h} + o(1)
\]  

(4)

\[
\frac{dp_k(t)}{dt} = -\lambda p_k(t) + \lambda \sum_{i=0}^{k-1} a_{k-i} p_i(t)
\]  

(5)

For \( k = 0 \), a special case of equation (5) is derived

\[
\frac{dp_0(t)}{dt} = -\lambda p_0(t)
\]  

(6)

The Erlang system of differential equations correspond to the formula (5) will have the following form

\[
\begin{align*}
\frac{df_i(t)}{dt} &= \lambda a_i f_{i-1}(t), \\
\frac{df_i(t)}{dt} &= \lambda \left( a_i f_i(t) + a_{i-1} f_{i-1}(t) \right), \\
\frac{df_i(t)}{dt} &= \lambda \left( a_i f_{i-1}(t) + a_{i-2} f_{i-2}(t) + \ldots + a_0 f_0(t) \right).
\end{align*}
\]  

(7)

where \( f_i(t) = p_i(t) e^{\lambda t} \), \( p_i(0) = f_i(0) = 0 \).

One way to solve the system of equations (7) is method based on the generating function usage [4]. In this case, we introduce a generalized notation for the generating function \( F(t, x) \) to define the probability function \( p_k(t) \)

\[
F(t + x) = \sum_{k=0}^{\infty} p_k(t) x^k
\]  

(8)

Then equation (5) and its special form equation (6) can be transformed by multiplying them by a factor \( x^k \) and summing over all \( k \), belonging to the half-interval \( k \in [0; \infty) \).
\[ \frac{dF}{dt} = -\lambda F + \lambda \sum_{k=1}^{\infty} x^k \sum_{i=1}^{\infty} a_i p_{k-i}(t) = \\
= -\lambda F + \lambda \sum_{i=1}^{\infty} a_i \sum_{j=0}^{\infty} p_j(t) x^{i+j} = -\lambda F + \lambda F(t,x) \sum_{i=1}^{\infty} a_i x^i \]

To obtain the desired generating function \( F(t,x) \) for a stationary flow without aftereffect, we introduce the following notation

\[ S(x) = \sum_{i=1}^{\infty} a_i x_i \]

Inasmuch as \( F(0,x) = p_0(0) = 1 \), define an expression for finding \( F(t,x) \)

\[ F(t,x) = e^{\lambda(S(x)-1)t} \]

To estimate the probability \( p_k(t) \) based on the generating function \( F(t,x) \), assume that we have several time slices that receive requirements and make up a stationary flow with intensity \( \lambda \). In this case, the probability \( a_k \) will be a characteristic of the \( k \) requirements appearance at the current time slice \( t \). Since the flow of requirements is stationary without aftereffect, the probability \( a_k \) will be unconditional, as we do not care about the contribution of previous requirements, as well as the order of requirements coming from processes handled by synchronization algorithms. Let us determine the probability \( p_k(t) \) of receiving requests \( k \) at a given time \( t \)

\[ p_k(t) = \sum_{j=1}^{\infty} \left( \frac{\lambda t^j}{j!} \right) p_j(k) \]

\[ F(x,t) = \sum_{k=0}^{\infty} P_j(k) x^k = (S(x))^j = \left( \sum_{i=1}^{\infty} a_i x^i \right)^j \]

where \( P_j(k) \) – probability of requirements occurrence \( k \) for time slices \( j \).

3. Representation of synchronization processes of distributed systems in the form of a queuing system model

To determine synchronization delays, as well as to find the probability of entering and exiting the synchronization state, it is necessary to determine the time distribution law of requests servicing. We suppose that a distributed synchronization system is a waiting \( QS \), where each thread waits for the resource, as long as it is owned by another process. In this paper, consider that we have a mixed-type waiting system, where the maximum waiting period for synchronization \( T_{\text{max}} \) is set for each processes, and the duration of servicing synchronization requests will be independent random variables that don’t depend on the input flow of requirements and have the same distribution law [5]. After the waiting period expires, the request to ensure synchronization is unprocessed, and the synchronization algorithm generates an exception. In this case, the waiting period can have a strictly defined value or be a random variable, which characteristics are determined depending on the workload of the computer system. In this case, the intensity of the requests outgoing flow will be equal to \( \mu = 1/M(T_{\text{avg}}) \), where \( T_{\text{avg}} \) – average service time [6]. It is assumed that the application service time has an exponential distribution of the following type
\[ \varphi(\sigma) = \begin{cases} 1 - e^{-\lambda t}, & \sigma \geq 0 \\ 0, & \sigma < 0 \end{cases} \]

where \( \sigma \) can take values \( \sigma_1, \sigma_2, \ldots, \sigma_n \) – the duration of servicing \( n \) types of requests received at the QS synchronization system input.

Introduced assumptions related to incoming demand flows, service times, and waiting times in the queue, allow us to use the mathematical apparatus of Markov random processes \( \xi(t) \) (chains) to model synchronization processes:

\[ P\left(\xi(t_{n+1}) = X_n | \xi(t_n) = X_n, \xi(t) = X_n\right) = P\left(\xi(t_{n+1}) = X_n | \xi(t) = X_n\right) \]

To describe such processes, it’s necessary to specify two types of probabilities: initial probabilities \( P\left(\xi(t_0) = j\right) = P_j(0) \) and conditional transition probabilities \( P\left(\xi(t_{n+1}) | \xi(t_n) = j\right) \). For homogeneous Markov processes, the probability \( P\left(\xi(t_{n+1}) | \xi(t_n) = j\right) \) depends only on adjacent time slices, thus another slices \( t = (t_1, t_2, \ldots, t_{n-1}) \) will not contribute to the transitions probability distribution from state \( t_n \) to \( t_{n+1} \). The transition probability for a homogeneous Markov process set as follows:

\[ P\left(\xi(t_{n+1}) = j | \xi(t_n) = i\right) = P_{ij}(\Delta t) = \sum_j P_{ij}(\Delta t) = 1 \forall \Delta t \]

where \( \Delta t = t_{n+1} - t_n \) – difference between time slices \( t_n \) and \( t_{n+1} \), \( i, j \) – the number of demands present in the system after operation completion of two adjacent demands for moments \( t_n \) and \( t_{n+1} \) respectively.

The simplest transition graph for the QS synchronization process with the incoming \( \lambda_m \) and outgoing \( \mu_m \) intensities of the request flows will have the following form:

![Figure 2. Markov process graph for QS synchronization.](image)

From Figure 2 it follows that \( n \) – describes the initial state of the system (a free channel), the transition into state \( n + 1 \) determines the entering to the critical section and completion synchronization for process \( Q_1 \), \( n + 2 \) the system has two processes \( Q_1 \) and \( Q_2 \) (for synchronizing \( Q_1 \) the \( Q_2 \) is in standby mode), \( n + m \) synchronization process execution for \( Q_m \) and the presence of threads \( Q = (Q_1, Q_2, \ldots, Q_{n+m}) \) waiting for synchronization. In this case, we get a condition that follows, when entering the critical section, synchronization for each process is performed only once. Note, that when an exception event occurs inside a critical section, the resource is released by the current process \( Q \) to avoid mutual locking [7]. This approach makes it possible to use Markov chains QS for modeling cyclic algorithms with synchronization. Let us consider that using the mathematical apparatus of Markov chains, transitions from state \( n \) to state \( n + 1 \) occur under the effect of a simple flow demands \( (P(t) = e^{-\lambda t}) \). Consider building a waiting Markov QS to implement the Suzuki-Kasami synchronization model displayed on the figure 1. To produce this, we write down the probabilities for marker transitions between processes waiting for synchronization in the form of the transition probabilities matrix.
The probabilities $P(\xi(t_0))$ and $P(\xi(t_{n+k})|\xi(t_n))$ satisfy the Chapman-Kolmogorov equation. The probabilities of states $P(\xi(t_{n+m}))$ as time-dependent functions from $t_{n+m}$ can be obtained from the Kolmogorov equation [8]. For this purpose, we introduce the following notation $P(t) = (P_{ij}(t))$ – the transition probabilities matrix for stochastic process $\xi(t)$. Let us write the Kolmogorov equations in the following form

$$\frac{dP(t)}{dt} = P(t)A(t), \quad \frac{dP(t)}{dt} = A(t)P(t),$$

(17)

where $I$ – identity matrix.

In equation (17), the multiplier $A(t) = a_{ij}$ is a matrix of infinitesimal transitions, formed from the derivatives, taken for each element of the transition matrix $P(t)$ at zero. The matrix of infinitesimal transitions has the following form [9]

$$A(t) = \begin{pmatrix} -\lambda & \lambda & 0 & 0 & \ldots \\ \mu & -(\lambda + \mu) & \lambda & 0 & \ldots \\ 0 & \mu & -(\lambda + \mu) & \lambda & \ldots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

(18)

The elements of the transition matrix $A(t)$ can be offer as a limit when $t \to 0$

$$a_{ii}(t) = \lim_{t \to 0} \frac{P_{ii}(t)}{t} < 0, \quad a_{ij}(t) = \lim_{t \to 0} \frac{P_{ij}(t)}{t} > 0, i \neq j$$

(19)

Taking into account the matrix of infinitesimal transitions, the Kolmogorov equations (17) for $P(t)$ can be rewritten in the following expanded form

$$\frac{dP_{ij}(t)}{dt} = \sum_{k=0}^{\infty} P_{ik}(t)a_{kj}(t),$$

$$\frac{dP_{ij}(t)}{dt} = \sum_{k=0}^{\infty} a_{ij}(t)P_{kj}(t), i,j = 0,1,\ldots,n$$

(20)

Note that when $t = 0$ we obtain a special case of the Kolmogorov equations (20)

$$P(0) = \begin{cases} 0, i \neq j \\ 1, i = j \end{cases}$$

(21)
In terms of the infinitesimal transitions matrix we obtain differential-difference equations of transition probabilities that characterize the process of death and propagation

\[
d\frac{P_x(t)}{dt} = \lambda P_{x-1}(t) - (\lambda + \mu) P_x(t) + \mu P_{x+1}(t), x = 1, 2, \ldots, n, \tag{22}
\]

To solve equation (22), we use the generating function \( F(S,t) \) and the Laplace transformation. The generating function corresponding to the probabilities \( P_x(t) \) have the following form

\[
F(S,t) = \sum_{x=0}^{\infty} P_x(t) S^x, |S| \leq 1. \tag{23}
\]

Using the expression for the generating function (23), we rewrite the expression (21):

\[
S \frac{dF(S,t)}{dt} = (1-S)((\mu-\lambda S)F(S,t) - \mu P_t(t)) \tag{24}
\]

Assume that the number of synchronization requests at a time \( t = 0 \) have a value \( X(0) = x_0 \), then assign the Laplace transformation \( f(S,t) = L(F(S,t)) \) for the generating function \( F(S,t) \). Given the constraints imposed \( X(0) = x_0 \), the generating function can be rewritten as \( F(S,0) = S^{x_0} \). Then the expression (24) can be reduced to the following form:

\[
f(S,z) = \frac{S^{x_0} - \mu(1-S)l(z)}{zS - (1-S)(\mu-\lambda S)}, \tag{25}
\]

where \( l(z) = L(P_0(t)) \) – Laplace transformation for \( P_0(t) \).

From the definition of the Laplace transformation, associated with function \( F(S,t) \), we obtain that \( f(S,z) \) will be finite at \( |S| \leq 1 \). Then in this area, the zero values of the numerator and denominator will be coincided:

\[
\delta_1 = \left(\frac{(\lambda + \mu + z) \mp (\lambda + \mu + z)^2 - 4\lambda\mu}{2\lambda}\right), \tag{26}
\]

\[
\delta_1 + \delta_2 = \frac{\lambda + \mu + z}{\lambda}, \delta_1 \times \delta_2 = \frac{\mu}{\lambda},
\]

where \( z = -\lambda(1-\delta)(1-\delta) \).

Applying Rouche’s theorem, we obtain that the denominator of expression (25) will take zero value only once inside the area \( |S| = 1 \). In this case, the expression for the generating function can be transformed by reducing on \( S - \delta \) and decomposing it into a series:
\[ f(S,z) = \frac{1}{\lambda} \sum_{i=0}^{\infty} \left( \frac{1}{\theta - \omega} - \frac{1}{\theta_{q_{i-r}+1} - \theta_{q_{i-r}+1}} + \frac{1}{\theta_{q_{i-r}+1} - \theta_{q_{i-r}+1}} \right) \times \sqrt{\omega^{q_{i-r}+1}} S^i \]

where \( \omega = \mu / \lambda, \delta_i = \sqrt{\omega} / \theta, \delta_i = \sqrt{\omega} \theta \).

We write the Laplace transform for generating function \( F(S,t) \) using the equation (24):

\[ L\left( \frac{dF(S,t)}{dt} \right) = zf(z,S) - S^\circ = -\lambda(1 - \delta_i)(1 - \delta_i) f(z,S) - S^\circ \]  

Substituting the Laplace transformation expression \( f(S,z) \) from expression (27) to formula (28), we can define the generating function expression \( P_x(t) \) as a multiplier parameter \( S^i \). It is easy to prove that the parameter \( L\left( \frac{dP_x(t)}{dt} \right) \) will be the sum of the first six terms of the series, each can be defined as the following relation:

\[ \frac{1}{\theta^{n+1}} + \frac{1}{\theta^{n-1}}, n \geq 0 \]  

The inverse Laplace transform for this parameter is written using the Bessel function \( I_n(x) \) in the following form

\[ \frac{1}{2\pi i} \int_{c-i\infty}^{c+i\infty} \frac{e^{zt}}{\theta^{n+1} - \theta^{n-1}} dz = \sqrt{\lambda \mu} \sqrt{e^{(\lambda + \mu)t}} \times I_n(2 \sqrt{\lambda \mu} t) \]  

Substituting the obtained relations into expression (21), we obtain the desired expression:

\[ \frac{dP_x(t)}{dt} = \sqrt{\frac{\mu}{\lambda} \lambda^{q_{i-r}}} e^{-(\lambda + \mu) t} - \left( \lambda + \mu \right) I_{q_{i-r}} + \sqrt{\lambda \mu} I_{q_{i-r}+1} + \right. \]
\[ + \sqrt{\lambda \mu} I_{q_{i-r}+1} + I_{q_{i+r}+2} + 2 \sqrt{\left( \lambda + \mu \right) I_{q_{i+r}+1} + \mu I_{q_{i+r}}} \],

where \( I_n = I_n(2 \sqrt{\lambda + \mu} t) \).

Note that from expression (31) it is possible to obtain a distribution for the queue length of process synchronization requests by integrating the right side of the expression.

4. Practical results

From a practical point of view, synchronization models based on QS have sufficient flexibility, allow to control the resources blocking process shared between different processes, and also reduce the risk of mutual locks. The Markov approach allows to evaluate the synchronization state transition for a certain process at any time, which makes it possible to control the procedure for sending tokens according to the Suzuki-Kasami model. In this paper, the Poisson flow was considered as an input stream, and in some situations, it is permissible to use other distributions. In this case, the mathematical apparatus of nested Markov chains can be used to calculate the probabilistic characteristics of the service discipline.

The main idea of the nested Markov chains method is to convert any random process that is not Markov to a Markov process. As a result, only those subprocesses whose characteristics can form a Markov chain are selected from the original random process. In this case, time slices are formed randomly depending on the parameters of the original process.
To evaluate the effectiveness of the proposed synchronization approach, we considered the possibility of using QS system for the Apache Spark parallel platform. This system is used to solve complex computational problems, in particular, the implementation of machine learning algorithms. We have considered the possibilities of built-in synchronization algorithms and made comparisons with the developed synchronization algorithm. It is worth noting that Apache Spark [10] uses a built-in broadcast mechanism that works on the BitTorrent protocol, which has basic capabilities for implementing broadcast procedures. This algorithm is based on the possibility of dividing blocks between individual nodes (processes). After the client connects to the master node that acts as a tracker, the addresses of other clients are transmitted to the client. After that, there is a persistent connection between clients and data exchange occur. In Spark, clients are parallel processes that initialize the procedure for accessing shared resources. This synchronization algorithm has one important drawback, due to the fact that in each process, data fragments of shared resources will accumulate, which requires the implementation of an additional mechanism for checking their integrity.

For getting experimental data, consider a typical Apache Spark cluster deployed using the KVM virtualization environment, with 10 nodes distributed on the Debian 10 operating system. As a test data, several files up to 1 GB in size are used, that uploaded into the distributed file system (Hadoop distributed system). Here is a comparison of the BitTorrent algorithm supplied in the standard Spark configuration and the synchronization algorithm based on the Suzuki-Kasami and QS model (SKQS).

### Table 1. Comparison of synchronization algorithms.

| Algorithm | Number of mutual locks per 1000 synchronization requests | Time synchronization of the data volume 100 MB | The maximum queue length processes |
|-----------|----------------------------------------------------------|---------------------------------------------|----------------------------------|
| BitTorrent | 10                                                       | 500 ms                                      | not limited                     |
| SKQS      | 0                                                        | 100 ms                                      | not limited                     |

Table 1 shows that the usage of SKQS algorithm reduces the threshold of mutual locks number, which allow to optimize the performance and fault tolerance of parallel algorithms executing in the Spark ecosystem. This factor is especially important when solving learning and probabilistic inference tasks, since if such exceptions occur, the learning process will either be completely stopped and the training must be repeated from the very beginning, or part of the training sample will not be taken into account in the formation of the initial probability distribution $P(X_0)$ and will lead to incorrect calculations based on such algorithms. The queuing theory usage makes it possible to evaluate the capabilities of a parallel system in terms of reliability and efficiency, as well as estimate the total volume of demands that can be processed by such a system. Evaluate the access waiting possibilities to shared resources during the execution of synchronization, as well as build probability distributions for the enter and exit state from the critical section for each parallel process. This approach allows to adapt parallel systems to solve machine learning problems, data classification and statistical analysis in the conditions of a large-scale input data.

5. Conclusions

Token transfer models, in particular the Suzuki-Kasami model, are universal tools for organizing synchronization process of distributed systems. However, with the current level of solving tasks complexity, optimization is required to reduce the number of exceptions and locks that occur during the transfer of tokens within the network. The application of queuing theory with combination of the Suzuki-Kasami algorithm makes it possible to deal these shortcomings by introducing probabilistic characteristics of the token transfer process, estimating the intensity of the demands input flow and time service characteristics associated with the waiting time processing distribution synchronization. Discrete Markov processes were used to calculate the characteristics of the QS. Calculations allow us to evaluate the capabilities of distributed synchronization system when processing resources with different data intensity and size. The developed system can be adapted to a wide range of incoming flow distributions
using the method of nested Markov chains or the method of decomposition of a random process into phases (the method of pseudo-states).

To accomplish experiment, a synchronization module for the Apache Spark parallel system was developed, which significantly expands the capabilities of this system, increases its resistance to exceptions and mutual locks, and allows to enhance the efficiency of system resource allocation in the process of executing parallel blocks and tasks with built-in synchronization.

References
[1] Suzuki I and Kasami T ACM Transaction on Computer Systems 3(4) 344-349
[2] Tanenbaum A S 2017 Distributed Systems: Principle and Paradigms (New York: Prentice Hall)
[3] Zinovyeva L I and Terpugov A F Automatic and Telemechanic 1 27-30
[4] Markushev A I 2006 Short Course Of Analytic Function Theory (Moscow: Mir)
[5] Toporkov V V 2004 Distributed system models (Moscow: Fizmatlit)
[6] Lifshitz A L 1978 Statistical Modelling Of Queue Systems (Moscow: Soviet Radio)
[7] Burns B 2018 Designing Distributed Systems: Patterns and Paradigms for Scalable Reliable Services (Sebastopol: O'Relly)
[8] Walrand J 1988 An Introduction To Queuing Networks (New Jersey: Prentice Hall)
[9] Klimov G 2011 Queue Systems (Moscow: Moscow University Publishing)
[10] Chowdhury M, Zaharia M Das, Ma J, Jordan M and Stoica I SIGCOMM 41(4) 98-109