DOI: 10.17725/rensit.2022.14.279

Problems of Message Routing Algorithms in Streaming Data Processing Systems

George G. Bulychev, Alexey V. Chernykh
MIREA-Russian Technological University, https://www.mirea.ru/
Moscow 119454, Russian Federation
E-mail: geo-bulychev@mail.ru, meidm@yandex.ru
Received May 26, 2022, peer-reviewed June 6, 2022, accepted June 13, 2022

Abstract: **Existing problems of routing algorithms, such as high overhead costs for scaling components and low fault tolerance of the system, are considered. An algorithm is proposed that eliminates the shortcomings of the existing ones due to the rejection of a centralized router and the strict separation of pipelines within the system. The algorithm is based on adding a special token to the message. For the proposed algorithm, the performance improvement is mathematically proved, and the practical load on the system is simulated to confirm the obtained theoretical results.**

Keywords: **streaming data processing, message routing, big data**

UDC 004.62

For citation: George G. Bulychev, Alexey V. Chernykh. Problems of Message Routing Algorithms in Streaming Data Processing Systems. *RENSIT: Radioelectronics. Nanosystems. Information Technologies*, 2022, 14(3):279-290. DOI: 10.17725/rensit.2022.14.279.

Contents

1. INTRODUCTION

The amount of data in the modern world is calculated in zettabytes [1] and every day their number is rapidly increasing. Such a volume of information opens up new and previously unseen opportunities for humanity to share knowledge and experience, statistical and marketing research, neural network training, early detection of threats, and much more. But in addition to opportunities, such volumes pose no less difficult tasks and challenges for researchers in implementing the mechanisms for collecting, processing and analyzing data.

The main approach currently used for big data analysis is the collection and aggregation of data into various storage systems with subsequent analysis. With this approach, the amount of resources consumed is proportional to the amount of data being processed. An alternative to this

approach is streaming data processing in real time, when a data processing pipeline is built and only the data that is directly needed to solve the task is stored. The most commonly used systems are ETL (Extract, Transform, Load) [2,3]. This class of systems consists of three categories of components: Extract – receiving data from external systems, Transform – data transformation, Load – transfer of processed data to third-party systems. Such systems are implemented in the form of pipelines, when data is transferred along the chain from one component to another.

For efficient data processing in real time, it is necessary to adapt data processing algorithms to work in the absence of storage facilities. It is necessary to develop universal algorithms [4] for solving a different range of problems, otherwise, for each specific study, it will be necessary to develop the required algorithms in the stream processing paradigm.

Separate requirements apply to the system itself. Each stage of processing can be performed on different equipment (including geo-distributed) to improve the efficiency of individual algorithms or data processing features related to the laws of various countries or the conditions for using information. The system should be centrally managed, with the ability to dynamically change the configuration of each individual component, and much more.

To solve all the problems described above, a streaming data processing system based on universal blocks is being developed. Such a system will allow implementing data processing scenarios in the shortest possible time by collecting chains of blocks through the user interface [5] and automatically launching them on any equipment.

One of the key aspects of the effectiveness of such systems is the message routing algorithm within the system. If the message processing path is not optimal, there may be unnecessary overhead for message passing, scaling individual components, or system fault tolerance. At the moment, the routing algorithms used in streaming data processing systems have significant drawbacks in terms of data processing efficiency or system fault tolerance.

2. ANALYSIS OF EXISTING ALGORITHMS

In data processing systems, only two algorithms are mainly used: with direct connection of components [6] and through a router [7].

2.1. Direct Join Algorithm

The direct connection algorithm creates two independent pipelines, within each of them all components interact directly with each other (**Fig. 1**).

If scaling is necessary, the system is able to create additional instances of a particular component within the same pipeline (**Fig. 2**). At the same time, since the pipelines are isolated, the scaling of the components within each pipeline will occur independently.

2.2. Router Algorithm

The router algorithm (**Fig. 3**), in turn, has one centralized component responsible for routing messages. Each component, except for the initiators, receives messages for processing

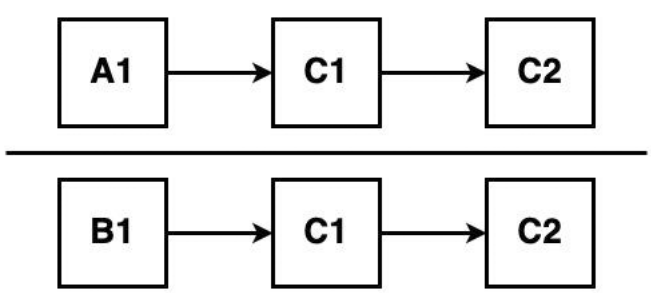

Fig. 1. *System architecture with direct connection algorithm.*

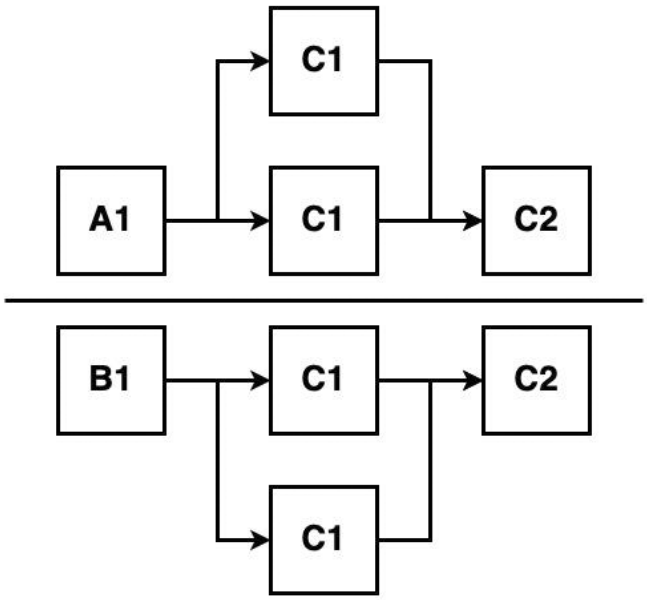

Fig. 2. *Component scaling in a system with direct component connection algorithm.*

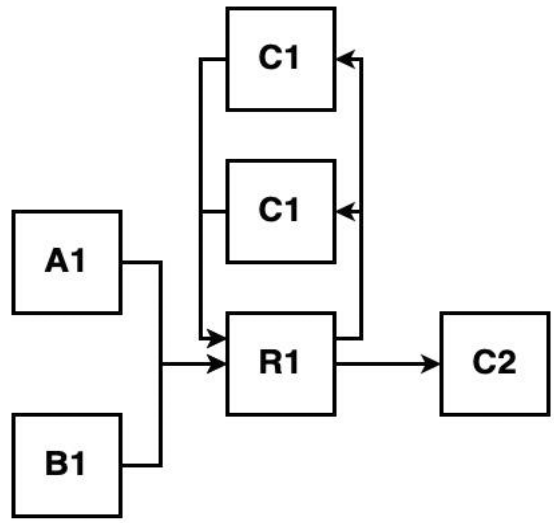

Fig. 4. *Component scaling in a system with a message routing algorithm through a router.*

from the central component, and after processing sends messages back to the router.

Since the essence of the pipeline becomes "virtual" in the algorithm, the components can be reused. In both pipelines, similar components perform the same task and have the same code base; components can process a message from one pipeline at one point in time, and from another at another point in time. The router, in turn, determines whether the message belongs to one of the pipelines and determines the further route of the message. Due to this property, the scaling of such components will depend on the total load from all pipelines (**Fig. 4**).

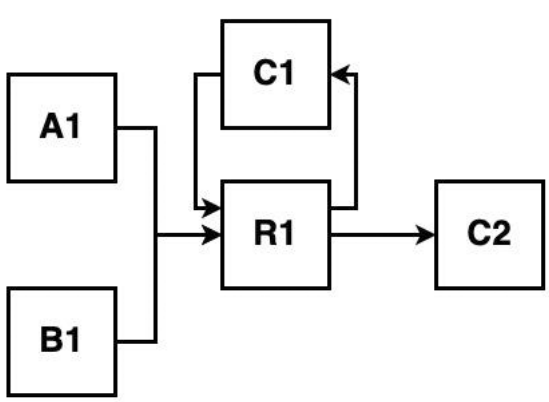

Fig. 3. *System architecture with an algorithm for routing messages through a router.*

2.3. Algorithm Comparison

Algorithms are compared according to six key parameters:

1. Number of message transfer operations between components.
2. The number of running component instances unique to each pipeline in the system.
3. The number of running instances of components reused in different pipelines in the system.
4. The number of operations for creating or deleting component instances during scaling.
5. Having a single point of failure.
6. Difficulty making changes to the pipeline.

Number of message transfer operations between components

In a direct connection algorithm, each message is passed directly from component to component, with no intermediate steps.

Therefore, the number of message transfer operations between components will be as follows.

$$C = n - 1, \quad (1)$$

where C – number of message transfer operations; n – number of components inside the conveyor.

In the algorithm with a router, the efficiency of using transmission channels between components is reduced, since after processing by a component, the message is sent to the router, and only after the router – to the next component. Therefore, the number of message transfer operations between components will be as follows.

$$C = 2(n - 1), \quad (2)$$

where C – number of message transfer operations; n – number of components inside the conveyor.

In the algorithm with a direct connection, the number of message transfer operations between components is two times less than in the algorithm with a router. Due to this, the overhead costs for the transmission of messages within the system are reduced.

Number of running instances of components unique to each pipeline in the system

Some components are unique for each pipeline, and accordingly, these components will be scaled only considering the load on a specific pipeline, regardless of the selected algorithm. If the system has N unique components, then the total number of all instances is expressed by the formula (3):

$$C_1 = \sum_{i=1}^{N} \left\lceil \frac{l_i}{p_i} \right\rceil, \quad (3)$$

where C_1 – number of unique component instances; N – number of unique components; l_i – load on the pipeline within which the component works; p_i – performance of a single component instance.

The number of running instances unique for each pipeline is the same in both algorithms.

Number of running instances of components reused in different pipelines in the system

In a direct connection algorithm, each pipeline is completely isolated, and the number of running instances of any of the components is determined by the load on a particular pipeline. Since reusable components are used in two pipelines, the total number of instances of a given component type is determined by the sum of the number of instances in each individual pipeline. If the system has **Z** pipelines and **N** components, then the total number of all instances is expressed by formula (4):

$$C_2 = \sum_{i=1}^{N} \sum_{j=1}^{Z} \left\lceil \frac{l_{ij}}{p_i} \right\rceil, \quad (4)$$

where C_2 – number of instances of reusable components; N – number of reusable components; Z – number of pipelines in the system; l_{ji} – load on a component in a pipeline; p_i – performance of a single component instance.

In the algorithm with a router, the number of running instances of reused components depends only on the total load on the system. Then the number of copies is expressed by the formula (5):

$$\tilde{N}_3 = \sum_{i=1}^{N} \left\lceil \sum_{j=1}^{Z} \frac{l_{ij}}{p_i} \right\rceil, \quad (5)$$

where C_3 – number of instances of reusable components; N – number of reusable components; Z – number of pipelines in the system; l_{ji} – load on a component in a pipeline; p_i – performance of a single component instance.

Since the sum of the ceilings is always greater than or equal to the ceiling of the sum, the following inequality is formed (6):

$$\sum_{i=1}^{N}\sum_{j=1}^{Z}\left[\frac{l_{ij}}{p_i}\right] \geq \sum_{i=1}^{N}\left[\sum_{j=1}^{Z}\frac{l_{ij}}{p_i}\right], \quad (6)$$

where N – number of reusable components; Z – number of pipelines in the system; l_{ji} – load on a component in a pipeline; p_i – performance of a single component instance.

The number of running instances of components reused in different pipelines in the router algorithm is less than or equal to that in the direct connection algorithm. Due to this, the efficiency of resource use in the algorithm with a router is higher. However, in the algorithm with a router, there is also an additional component - the router.

Number of operations to create or remove component instances when scaling them

To compare the two algorithms, consider the time interval on which the function of changing the load on conveyors monotonically increases or decreases. Since the number of unique component instances is the same in both algorithms, only reusable components will be considered.

The number of operations for creating or deleting component instances in a direct routed algorithm will be (7):

$$C_4 = \sum_{i=1}^{N}\sum_{j=1}^{Z}\left[\frac{|lend_{ij} - lstart_{ij}|}{p_i}\right], \quad (7)$$

where C_4 – number of instance creation or deletion operations; N – number of reusable components; Z – number of pipelines in the system; $lend_{ji}$ (load end_{ij}) – load on the component in the pipeline at the end of the time interval; $lstart_{ji}$ (load $start_{ij}$) – load on the component in the pipeline at the beginning of the time interval; p_i – performance of a single component instance.

The number of operations for creating or deleting component instances in the algorithm with a router will be (8):

$$C_5 = \sum_{i=1}^{N}\left[\sum_{j=1}^{Z}\frac{|lend_{ij} - lstart_{ij}|}{p_i}\right], \quad (8)$$

where C_5 – number of instance creation or deletion operations; N – number of reusable components; Z – number of pipelines in the system; $lend_{ji}$ (load end_{ij}) – load on the component in the pipeline at the end of the time interval; $lstart_{ji}$ (load $start_{ij}$) - load on the component in the pipeline at the beginning of the time interval; p_i – performance of a single component instance.

Since the sum of the ceilings is always greater than or equal to the ceiling of the sum, the following inequality is formed (9):

$$\sum_{i=1}^{N}\sum_{j=1}^{Z}\left[\frac{|lend_{ij} - lstart_{ij}|}{p_i}\right] \geq \sum_{i=1}^{N}\left[\sum_{j=1}^{Z}\frac{|lend_{ij} - lstart_{ij}|}{p_i}\right], \quad (9)$$

where N – number of reusable components; Z – number of pipelines in the system; $lend_{ji}$ (load end_{ij}) – load on the component in the pipeline at the end of the time interval; $lstart_{ji}$ (load $start_{ij}$) – load on the component in the pipeline at the beginning of the time interval; p_i – performance of a single component instance.

The number of operations for creating or deleting component instances during scaling in the routing algorithm is less than or equal to the number of operations in the direct connection algorithm. This results in lower component scaling overhead.

Having a single point of failure

In a direct connection algorithm, due to the independence of pipelines, there is no single point of failure for the entire system.

In the router algorithm, there is a central component responsible for routing packets throughout the system. If this component fails, the entire system will fail.

Difficulty making changes to the pipeline

In a direct connection algorithm, the pipeline is formed at the initialization stage. To make the change, you must completely reinitialize the pipeline, which will incur additional overhead.

In the router algorithm, the pipeline is formed based on the routing table, which is stored in the central component. To make changes to the pipeline, it is enough to change the routing table.

Conclusions from the comparison of algorithms

The direct connection algorithm has a high level of fault tolerance and resource efficiency for processing a single message, since the message is transmitted along the optimal route. However, the resource efficiency of the entire system is reduced by the inability to reuse components in multiple pipelines.

In turn, the algorithm with a router uses the resources of the entire system more efficiently, due to the reuse of components, however, fault tolerance is significantly reduced, and additional overhead costs for message transmission between components arise.

Algorithms have their pros and cons, however, to improve the efficiency of resource use, a new algorithm is needed that combines the key advantages of the presented algorithms and levels their shortcomings.

3. ALGORITHM FOR ROUTING MESSAGES USING TOKENS

It's necessary to achieve the transfer of messages along the optimal route, while maintaining the possibility of reusing components in different pipelines.

To solve this problem, it is proposed to abandon the use of a central router in favor of additional modules integrated into components. The message marking module is integrated into the first component. The marker indicates that the message belongs to a particular pipeline. A routing module is integrated into the processing components, which is responsible for determining the further route of the message based on the token (**Fig. 5**).

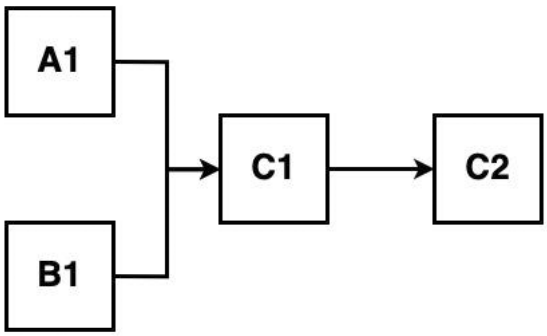

Fig. 5. *System architecture with message routing algorithm using tokens.*

When the load on the entire system increases, individual components are scaled up (**Fig. 6**). When the load drops – the number of instances of individual components will decrease.

Thanks to this architecture, the efficiency of using system resources will be approximately the same as in the case of the router algorithm, since the number of component instances will depend on the load on the entire system due to the reuse of components. However, due to the additional module, the performance of the components will be reduced. The number of component instances will be expressed by the formula (10):

$$C_6 = \sum_{i=1}^{N}\left[\sum_{j=1}^{Z}\frac{l_{ij}}{p_i - pr}\right], \qquad (10)$$

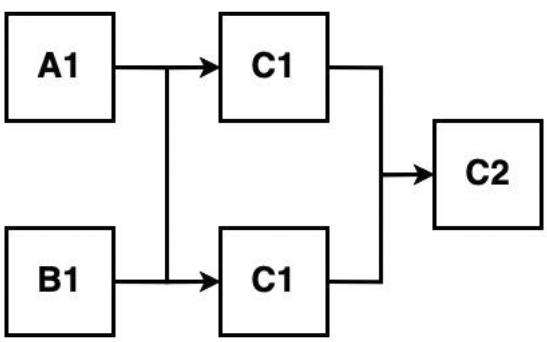

Fig. 6. *Scaling of "C_1" in a system with message routing algorithm using markers.*

where C_3 – number of instances of reusable components; N – number of reusable components; Z – number of pipelines in the system; l_{ji} – load on a component in a pipeline; p_i – performance of a single component instance; p_r – performance of the routing module in the component.

The resulting difference in the number of components will directly depend on the actual implementation of the routing module.

The efficiency of using message passing channels between components will be similar to the direct connection algorithm, since messages are transferred directly from component to component within the pipeline route. Since the routing module is integrated directly into each component, the cost of sending a message to the module will be significantly lower than sending it over a network link to the central routing component. Therefore, the number of message transfer operations between components will be as follows.

$$C = n - 1, \quad (11)$$

where C – number of message transfer operations; n – number of components inside the conveyor.

Due to the absence of a central component, the fault tolerance of the system will be similar to the algorithm with a direct connection of components.

However, a way to route the message based on the token needs to be developed.

Similar algorithms are used in network packet routing systems through the use of a routing table [8]. When applying such an algorithm to a streaming data processing system, it is necessary to keep the routing table in the router of each component up to date. Then the router can use the message token to find information in the routing table about which pipeline the given message belongs to and the current position of the message relative to the route of the pipeline. Depending on the route, the message is sent to the next component (**Fig. 7**).

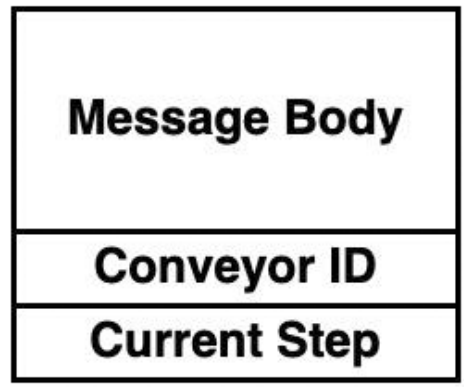

Fig. 7. *Message structure with marker.*

The main disadvantage of this approach is the need to keep the routing tables in each component up to date.

An alternative is to consider special markers, inside which the entire message route will be located (**Fig. 8**). When the router inside the component receives essages, it retrieves the route and the current position of the message relative to the route from the marker. Depending on the route, the message is passed to the next component.

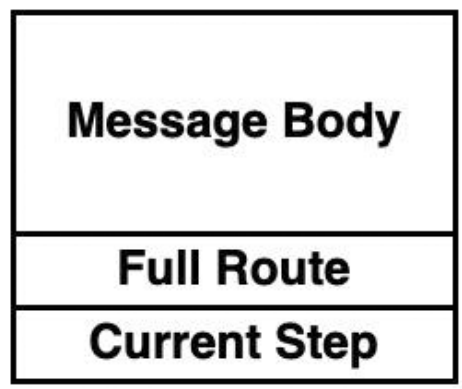

Fig. 8. *Message structure with route.*

Such message routing algorithms within a streaming data processing system require the introduction of an additional module in the first component of each pipeline, which is responsible for marking the message with a marker. The main disadvantage of the algorithm with a marker will be leveled, since it is necessary to synchronize routes with only one component inside each pipeline, and not with all.

4. SIMULATION ALGORITHMS

Consider a system consisting of two conveyors. The Alpha pipeline (**Fig. 9**) consists of three components: A1 – receiving messages from the "AlphaNet" social network, C1 – evaluating the emotional coloring of the message, C2 – saving messages to the database.

The Betta pipeline (**Fig. 10**) consists of three components: B1 – receiving messages from the "BettaNet" social network, C1 – evaluating the emotional coloring of the message, and C2 – saving messages to the database.

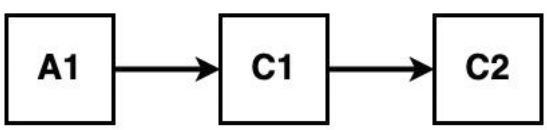

Fig. 9. *Alpha pipeline.*

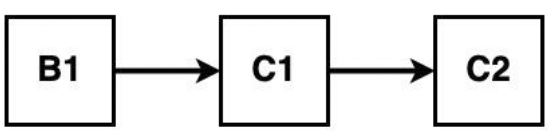

Fig. 10. *Betta pipeline.*

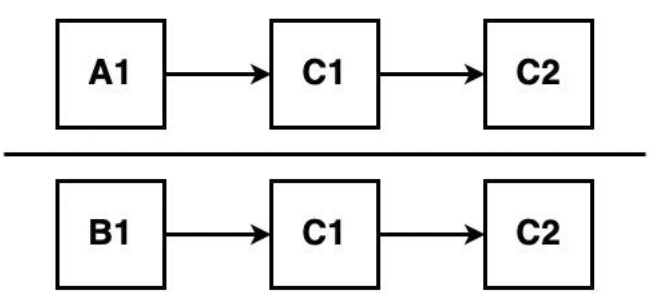

Fig. 11. *Simulation system architecture with direct connection algorithm.*

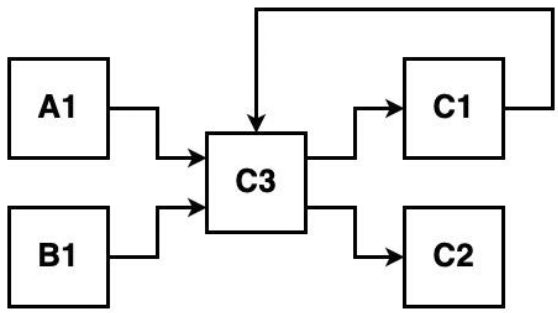

Fig. 12. *Simulation system architecture with an algorithm for routing messages through a router.*

Components A1 and B1 are unique for each conveyor. Components C1 and C2 are the same in the pipelines.

If you use the algorithm with direct connection of components, you will get the following system (**Fig. 11**).

When using the algorithm with the router, the following system is formed (**Fig. 12**). Also a new component C3 – message router – is added to the system.

If you use the routing algorithm with message labeling, you get the following system (**Fig. 13**).

The performance of each component is shown in **Table 1.**

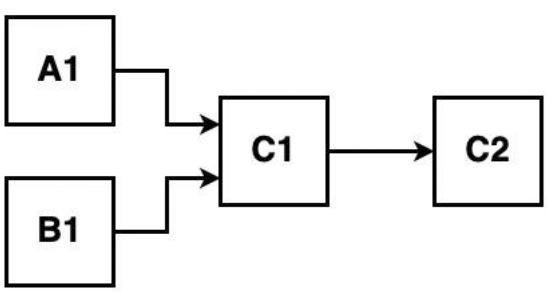

Fig. 13. *Simulation system architecture with message routing algorithm using tokens.*

Table 1

Component performance.

Component	Perfomance (msg./sec.)
A1	30000
B1	40000
C1	9000
C2	55000
C3	90000

Table 2

Load on conveyors.

Time	Alpha load (msg./sec.))	Betta Load (msg./sec.)
1	100000	0
2	180000	30000
3	180000	100000
4	120000	140000
5	80000	180000
6	40000	230000
7	20000	280000
8	40000	200000
9	100000	180000
10	150000	120000
11	200000	80000

Since in a routing algorithm with message tagging the performance of components may decrease, due to the need to tag messages or determine the further route of messages, let us introduce a performance degradation factor.

$$K = 0.95, \quad (12)$$

where, K – degradation factor

Each conveyor has a unique load applied to it. The load changes every hour. An increase in the number of instances of a component occurs when the maximum load that an instance can handle is reached. A decrease in the number of instances of a component occurs when the load on an instance decreases to zero.

The load on each conveyor for 11 hours is shown in **Table 2.**

4.1. Simulation of an algorithm with direct connection

In a system with a direct component linkage algorithm, the number of instances of each component depends only on the load on the component in a particular pipeline, the final number of component instances C1 and C2 is calculated as the sum of the component instances in each pipeline.

The number of components of each type is shown in **Table 3.**

There were 464 instances of a component in the system during the whole time of operation.

Table 3

The number of instances of components in a system with direct connection.

Time	Alpha load (msg./ sec.)	Betta load (msg./ sec.)	Number of A1	Number of B1	Number of C1	Number of C2
1	100000	0	3	0	12	2
2	180000	30000	5	1	24	5
3	180000	100000	5	4	32	6
4	120000	140000	3	5	30	6
5	80000	180000	2	6	29	6
6	40000	230000	1	8	31	6
7	20000	280000	1	10	35	7
8	40000	200000	1	7	28	5
9	100000	180000	3	6	32	6
10	150000	120000	4	4	31	6
11	200000	80000	5	3	32	6

Table 4 shows the number of component instances created or deleted.

There have been 162 operations to create or delete component instances over the lifetime of the system.

4.2. Simulating an algorithm with a router

In a system with a router algorithm, the number of instances of each component depends only on the total load on the component. There is also a C3 router in the system.

Table 4

The number of operations in a system with direct connection of components.

Time	Alpha load (msg./ sec.)	Betta load (msg./ sec.)	Operations A1	Operations B1	Operations C1	Operations C2
1	100000	0	3	0	12	2
2	180000	30000	2	1	12	3
3	180000	100000	0	3	8	1
4	120000	140000	2	1	10	2
5	80000	180000	1	1	9	2
6	40000	230000	1	2	10	2
7	20000	280000	0	2	8	1
8	40000	200000	0	3	11	2
9	100000	180000	2	1	10	1
10	150000	120000	1	2	11	2
11	200000	80000	1	1	11	2

Table 5

The number of component instances in a system with a router.

Time	Alpha load (msg./ sec.)	Betta load (msg./ sec.)	Number of A1	Number of B1	Number of C1	Number of C2	Number of C3
1	100000	0	3	0	12	2	2
2	180000	30000	5	1	24	4	3
3	180000	100000	5	4	32	6	4
4	120000	140000	3	5	29	5	3
5	80000	180000	2	6	29	5	3
6	40000	230000	1	8	30	5	3
7	20000	280000	1	10	34	6	4
8	40000	200000	1	7	27	5	3
9	100000	180000	3	6	32	6	4
10	150000	120000	4	4	30	5	3
11	200000	80000	5	3	32	6	4

The number of components of each type is shown in **Table 5**.

There have been 489 instances of a component in the system over the lifetime of the system.

Table 6 shows the number of component instances created or deleted.

There have been 108 component instance creation or deletion operations in the system over the lifetime of the system.

Table 6

The number of operations in a system with a router.

Time	Alpha load (msg./ sec.)	Betta load (msg./ sec.)	Operations A1	Operations B1	Operations C1	Operations C2	Operations C3
1	100000	0	3	0	12	2	2
2	180000	30000	2	1	12	2	1
3	180000	100000	0	3	8	2	1
4	120000	140000	2	1	3	1	1
5	80000	180000	1	1	0	0	0
6	40000	230000	1	2	1	0	0
7	20000	280000	0	2	4	1	1
8	40000	200000	0	3	7	1	1
9	100000	180000	2	1	5	1	1
10	150000	120000	1	2	2	1	1
11	200000	80000	1	1	2	1	1

Table 7

The number of components instances in a system with message tagging.

Time	Alpha load (msg./ sec.)	Betta load (msg./ sec.)	Number of A1	Number of B1	Number of C1	Number of C2
1	100000	0	3	0	12	2
2	180000	30000	5	2	25	5
3	180000	100000	5	4	33	6
4	120000	140000	4	5	31	5
5	80000	180000	3	7	31	5
6	40000	230000	2	9	32	6
7	20000	280000	1	10	36	6
8	40000	200000	2	8	29	5
9	100000	180000	3	7	33	6
10	150000	120000	4	5	32	6
11	200000	80000	6	3	33	6

4.3. Simulation an algorithm with message tagging

In a system with a message-labeled algorithm, the performance of each component is reduced. The number of instances of each component depends only on the total load on the component.

The number of components of each type is shown in **Table 7**.

There have been 483 instances of a component in the system over the lifetime of the system.

Table 8 shows the number of component instances created or deleted.

Table 8

The number of operations in a system with message tagging.

Time	Alpha load (msg./ sec.)	Betta load (msg./ sec.)	Operations A1	Operations B1	Operations C1	Operations C2
1	100000	0	3	0	12	2
2	180000	30000	2	2	13	3
3	180000	100000	0	2	8	1
4	120000	140000	1	1	2	1
5	80000	180000	1	2	0	0
6	40000	230000	1	2	1	1
7	20000	280000	1	1	4	0
8	40000	200000	1	2	7	1
9	100000	180000	1	1	4	1
10	150000	120000	1	2	1	0
11	200000	80000	2	2	1	0

There have been 94 operations to create or delete component instances over the lifetime of the system.

4.4. Simulation Findings

The lowest total number of component instances in the system over all time is generated in the algorithm with direct connection - 464, against 489 in the algorithm with router and 483 in the algorithm with message tagging. The difference is due to the need to scale an additional component in the router algorithm and the reduced component performance in the message tagging algorithm. However, the difference with the message labeling algorithm is 4.09%.

The message tagging algorithm has the lowest number of operations to create or remove instances of components - 94 operations, compared to 162 operations for the direct join algorithm and 108 for the router algorithm. The difference is 41.98% with the direct join algorithm.

An algorithm with message tagging shows better results in terms of reducing the number of component creation or deletion operations than an algorithm with a router or a direct connection. Depending on the implementation, the performance reduction factor may vary, and the identified difference in the number of instances may be evened out.

5. CONCLUSION

The proposed message routing algorithm using a token eliminates the main disadvantages of previously used algorithms, such as: reduced efficiency of using system resources in the direct routing algorithm, low fault tolerance and efficient use of message transmission channels in the algorithm with a router.

However, the message marking algorithm will require additional resources for routing and an increased message size, in contrast to the direct connection algorithm, and the final efficiency of use will depend on the optimal implementation of the routing module, without restrictions from the algorithms.

Thanks to the developed algorithm, the possibility of analyzing large amounts of data increases, the fault tolerance of the system increases, and the cost of the resources used is reduced.

However, in this article, only the classic scaling algorithm was considered - increasing or decreasing the number of component instances depending on the current load. However, this algorithm may be inefficient if the function is not monotone on the time interval. Optimization of the scaling algorithm will improve system performance by reducing the overhead costs for creating and deleting component instances.

REFERENCES

1. David Reinsel, John Rydning, John F. Gantz. IDC's Global DataSphere Forecast Shows Continued Steady Growth in the Creation and Consumption of Data. https://www.idc.com/getdoc.jsp?containerId=prUS46286020.
2. David Loshin. *ETL (Extract, Transform, Load). Business Intelligence: the Savvy Manager's Guide*, 2nd Edition. Elsevier, 2012, 400 p.
3. Vassiliadis P, Karagiannis A, Tziovara V, Simitsis A. Towards a Benchmark for ETL Workflows. *5th International Workshop on Quality in Databases,* 2007, 49-60 pp.
4. Simitsis A, Vassiliadis PA. A method for the mapping of conceptual designs to logical blueprints for ETL processes. *Decision Support Systems*, 2008, 45(1):22-40. DOI:10.1016/j.dss.2006.12.002.
5. Skoutas D, Simitsis A. Designing ETL processes using semantic web technologies. *Proceedings ACM 9th International Workshop on Data Warehousing and OLAP*, 2006, 67-74 pp.

6. Alkis Simitsis, Panos Vassiliadis, Timos Sellis. Optimizing ETL processes in data warehouses. *Proceedings of the International Conference on DataEngineering (ICDE)*, 2005.
7. Camunda Platform 7: Architecture Overview. URL: https://docs.camunda.org/manual /7.16/introduction/architecture/ (Accessed 04/12/2022).
8. Baker F. *RFC 1812 Requirements for IP Version 4 Routers*. URL: https://datatracker.ietf.org/doc/html/rfc1812 (Accessed 04/12/2022).