# Extract-Load-Transform (ELT) Process Runtime Analysis and Optimization

2023 5th International Youth Conference on Radio Electronics, Electrical and Power Engineering (REEPE) | 979-8-3503-9952-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/REEPE57272.2023.10086728

Aleksei E. Zvonarev[1], Dmitriy S. Gudilin[2], Dmitriy A. Lychagin[3], Boris S. Goryachkin[4]
*Bauman Moscow State Technical University*
Moscow, *Russian Federation*
defygee@yandex.ru[1], dima.gudilin@mail.ru[2], lychagin.dmitry@gmail.com[3], bsgor@mail.ru[4]

***Abstract*—The article discusses algorithms for optimizing the transformation stage of the ELT process, built on the basis of procedures in the PostgreSQL DBMS and the parallelization mechanism implemented by the Python programming language tools. As a basis for comparison, the most trivial version of the data conversion process was taken, which consists in a sequential connection of each individual procedure. The first proposed algorithm uses the principle of the simplest parallelization of procedures, which allows you to perform independent procedures in parallel. The second algorithm is an improved version of the first one. It uses the principle of step-by-step optimization with additional parallelization of chain blocks of dependent procedures. As the main criterion for evaluation, the time of execution of the entire chain of procedures was taken. As a result of the study, it was determined that the improved version of the procedure parallelization algorithm shows the shortest execution time of the entire chain of the data transformation step.**

***Keywords— optimization, ELT, Python, PostgreSQL, flow, procedure, DBMS***

## I. Introduction

Within organizations, data is often scattered across different systems [1] and rendered useless by not being centrally stored in the right system and format [2]. It is hard to discover [3] all the useful insights from this data. Extraction, Loading and Transformation (ELT) is a method currently used to transfer data from the source system to the data store. It involves retrieving data from many external sources, loading it into a target database (DB) or data store [4], and then converting it to meet operational needs (sometimes using intermediate tables) [5]. During the extraction process, data is extracted from various data sources for further processing or storage. Data Transformation is responsible for converting a set of data values from the source system format to the target system format according to the business logic provided by the customer in the form of a business requirements document [6].

This article will discuss the analytical approach of building the architecture of the ELT process transformation stage using the PostgreSQL DBMS and the Python programming language and built an algorithm to solve the problem of its sequential optimization.

## II. Literature Review

Not as many methods have been proposed in the literature for optimizing ELT processes as for ETL processes, but many approaches and methods for the data transformation phase using parallelization can be reused in ELT processes. In [7] the author considers several different options for optimizing ETL processes, such as: State-space approach, Dependency graph, Scheduling strategies, Reusable patterns, Parallelism, Quality metrics, Statistics, the use of Commercial ETL tools.

In [8] a methodology for paralleling processes in the ELT workflow is also presented, but this work also lacks specific optimization algorithms.

Also, there are several ETL semantic frameworks [9] [10] in literature which focuses on providing semantics to various data fields thereby facilitating richer data integration. This approach could be useful but cannot produce ELT architecture.

Our work reveals in more detail the variants of algorithms for step-by-step parallelization of procedures related to the Parallelism part in [7] and the methodology in [8], using the example of PostgreSQL DBMS and tools written in the Python programming language.

## III. Description of the ELT Process

Standard ELT process scheme includes a number of source systems from which data must be transferred to a single data warehouse [9] on an ongoing basis. After data is collected in a single repository, it can be transformed to solve various types of business or technical problems. ELT process diagram is shown in Figure 1.

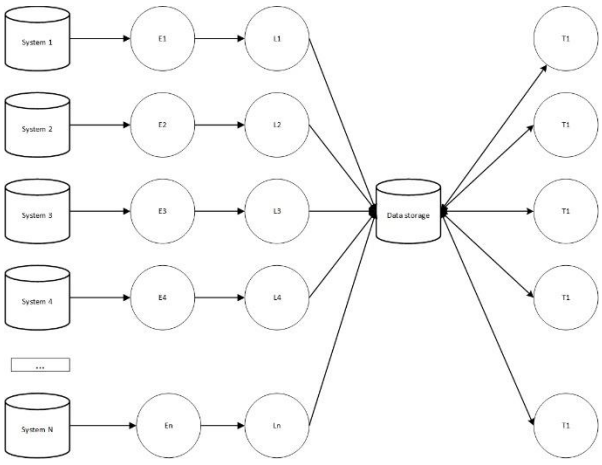


Fig. 1. ELT Process Diagram.

As example of processes, that occur during the data transformation phase, a part of the stage of the final transformation of the ELT process, developed to build the analytical reporting [11] of the bank [12], is taken. This stage consists of 14 PostgreSQL procedures [13].

The initial and most trivial solution will be to perform [14] this set of procedures sequentially. This graph is shown in Figure 2.

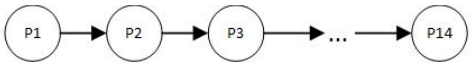


Fig. 2. Set of procedures.

To estimate the time of sequential execution of this set of procedures, we need to use the following formula:

$$T_{agg} = \sum_{i=1}^{n} P_i = t_{p1} + t_{p2} + t_{p3} + \dots + t_{pn} \quad (1)$$

Experimental time of each procedure tpn is shown is the Table I:

TABLE I. PROCEDURE EXECUTION TIME

| Procedure | Run Time (seconds) |
|---|---|
| 1 | 30 |
| P2 | 120 |
| P3 | 90 |
| P4 | 110 |
| P5 | 60 |
| P6 | 70 |
| P7 | 100 |
| P8 | 76 |
| P9 | 194 |
| P10 | 181 |
| P11 | 531 |
| P12 | 11 |
| P13 | 189 |
| P14 | 300 |

Using the formula obtained earlier, the total time of the entire process will be:

$$T_{agg} = \sum_{i=1}^{n} P_i = \sum_{i=1}^{14} P_i = 2062\ seconds$$

## IV. ELT PROCESS OPTIMIZATION BY PARALLELIZATION

First of all, to optimize the ELT process, it is necessary to parallelize the execution of individual process chains. For example, it is proposed to consider two options for performing consecutive chains: sequential and parallelized.

As example, the following set of procedures presented in Table II is taken. Dependencies between procedures are shown in Figure 3.

TABLE II. EXAMPLE OF POSTGRESQL PROCEDURES

| *Procedure* | *Run Time (seconds)* |
|---|---|
| P1 | 60 |
| P2 | 42 |
| P3 | 66 |
| P4 | 66 |
| P5 | 70 |
| P6 | 20 |

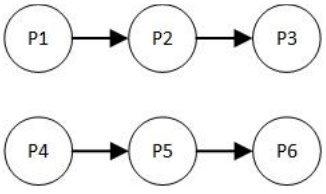


Fig. 3. Example of dependencies between procedures.

When performed sequentially, the total time will be taken by formula (1) and will be equal to:

$$T_{agg} = \sum_{i=1}^{6} P_i = t_{p1} + t_{p2} + t_{p3} + t_{p4} + t_{p5} + t_{p6} = 60 + 42 + 66 + 66 + 70 + 20 = 324 \text{ seconds}$$

When parallelizing two independent chains, the execution time will be calculated using the following formula:

$$T_{agg} = \max\left(\sum_{i=1}^{n1} P_i,\ \sum_{i=n1+1}^{n2} P_i\right) \quad (2)$$

Calculation of the runtime for this example will be equal to 128 seconds.

In this case, the parallel execution time of the request is almost 3 times less than the sequential time, which shows, that in order to optimize the execution time of ELT processes, it is necessary to use the method of parallelizing independent process chains.

## V. ELT PROCESS TIME OPTIMIZATION ALGORITHM

PostgreSQL does not have the necessary functionality to perform [15] this task, and therefore it is required to use additional software.

As the simplest option for solving this problem, it is proposed to use the tools of the high-level Python programming language and its Threading library.

Threading is a standard module that comes with an interpreter. The programmer does not need to install it, just connect the module using the command: import threading.

It is possible to work with threads by creating instances of the Thread class. To create a separate thread, it is required to create an instance of the class and apply the start() method to it.

Using these tools makes it possible to parallelize procedures [16] written in PostgreSQL.

## VI. PROCEDURE PARALLELIZATION ALGORITHM

***Step 1***. Create an interaction matrix between procedures and tables.

Each of the procedures executes a specific set of algorithms, while interacting with a number of tables. All existing relationships between tables must be identified. To define a set of relationships, it is required create a mapping matrix between the tables used and the SQL statements that execute on them.

***Step 2***. Compiling Dependency Sets Between Tables.

This step is preparatory for step 3. It is required to draw dependencies between tables in the format: $T_a$ -> $T_b$ $T_c$ $T_d$. It is possible to get a set of dependency data from the interaction matrix built on step 1. The dependency is determined by the following rule:

***Rule 1***. If in the procedure Pa the insert, update, or delete action is performed on the table $T_a$ and a select action for the table $T_b$, then table $T_a$ will be a dependent table on $T_b$ and this link will be recorded as follows: $T_b$ -> $T_a$.

***Step 3***. Compiling Dependency Sets Between Procedures.

Based on the set of dependencies between tables obtained at step 2 and the interaction matrices obtained at step 1, it is required to create dependency sets between procedures. This is done according to the following rule:

***Rule 2***. If in the procedure Pa the insert, update, or delete action is performed on the table $T_a$, then all the procedures in which for the table $T_a$ use select action will be procedure dependent $P_a$.

***Step 4***. Iterative optimization of the ELT process by creating threads.

After obtaining a set of dependencies between procedures, it is required to iteratively compose threads. In order to do this, it is necessary at each step to write out those procedures that are not dependent on any of the sets obtained at step 3. All procedures written by such a mechanism will go to a flow which number corresponds to the iteration number. All procedures in the same thread will run in parallel.

Example of dependencies between procedures is shown in Table III:

TABLE III. PROCEDURE EXECUTION TIME

| *Procedure* | *insert* | *update* | *delete* | *select* |
|---|---|---|---|---|
| P1 | T2 | | | T1 |
| P2 | T3, T4 | T3 | T3 | T2 |
| P3 | T5 | | | T3, T4 |
| P4 | T6 | | | T2 |
| P5 | T7 | | | T5 |
| P6 | T8 | | | T1, T2, T5, T6 |
| P7 | T9 | | | T3, T4, T7, T8 |
| P8 | T11 | | T11 | T6 |
| P9 | T10 | | | T12, T13 |
| P10 | T13 | T13 | | T14 |
| P11 | T9 | | | T15, T10, T11 |
| P12 | T9 | | T11 | T11 |
| P13 | T9 | | | T10 |
| P14 | T15 | | | T7 |

The select statement for the final data mart must be executed after the insert, update, and delete are completed. Also, insert, update and delete statements cannot be executed in parallel due to the likelihood of data integrity violations. Relationships for tables are:

T1 -> T2, T8; T2 -> T3, T4, T6, T8; T3 -> T5, T9; T4 -> T5, T9; T5 -> T7, T8; T6 -> T8, T11; T7 -> T9, T15; T8 -> T9; T9 -> {}; T10 -> T9; T11 -> T9; T12 -> T10; T13 -> T10; T14 -> T13; T15 -> T9

Based on this data, it is possible to build links between procedures:

P1-> P2, P4, P6; P2-> P3, P7; P3 -> P5, P6; P4-> P6, P8; P5-> P7, P14; P6-> P7; P7-> {}; P8-> P11, P12; P9-> P11, P13; P10-> P9; P11-> P7; P12-> P7; P13-> P7; P14-> P11

To find the procedures to be performed in the first thread, it is necessary to select those that are not on the right side of the relationship between the procedures, such procedures are P1 and P10. These procedures will go to the first flow.

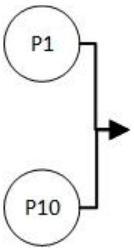


Fig. 4. First flow.

Removing them from set will lead to the following:

P2-> P3, P7; P3 -> P5, P6; P4-> P6, P8; P5-> P7, P14; P6-> P7; P7-> {}; P8-> P11, P12; P9-> P11, P13; P11-> P7; P12-> P7; P13-> P7; P14-> P11

Similarly, it is required to select those that are not on the right side to select procedures in the second thread. Such procedures are P2, P4 and P9. They will be added to the second flow.

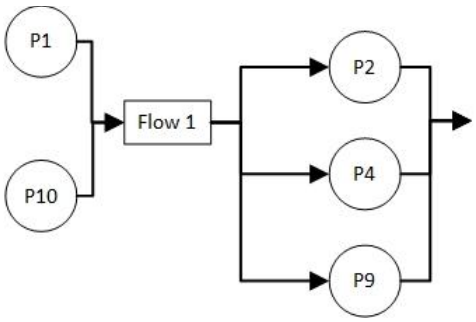


Fig. 5. Second flow.

After removing them from set such result is obtained:

P3 -> P5, P6; P5-> P7, P14; P6-> P7; P7-> {}; P8-> P11, P12; P11-> P7; P12-> P7; P13-> P7; P14-> P11

P3, P8 and P13 are not on the right side. They will be added to the third flow.

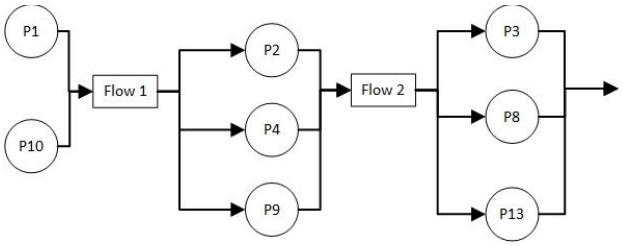


Fig. 6. Third flow.

Next iteration:

P5-> P7, P14; P6-> P7; P7-> {}; P11-> P7; P12-> P7; P14-> P11

P12, P5 and P6 are not on the right side. They will be added into the flow 4.

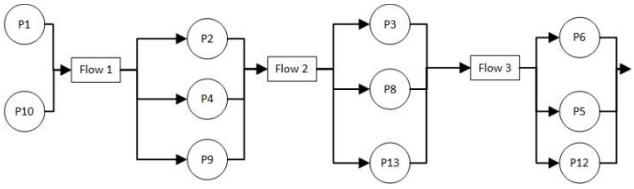


Fig. 7. Fifth flow.

Next iteration:

P7-> {}; P11-> P7

P7 depends on the execution P11. P11 will be added to the sixth flow, and P7 to the seventh. The final chain is shown in Figure 8.

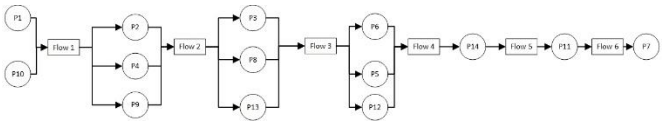


Fig. 8. Final chain.

Substituting in formula 1, the runtime before optimization and after is obtained:

$$T_{agg} = \sum_{i=1}^{n} P_i = \sum_{i=1}^{14} P_i = 2062 \text{ seconds}$$

The total time of parallel and sequential operation will also be considered by formula 1, but instead of the time of execution of a separate procedure, it is necessary to take the time of execution of the thread. To find the operating time of each stream, use formula 2. As a result, it will be equal to 1376 seconds

## VII. Improved Procedure Parallelization Algorithm

For more complex chains of dependencies, an algorithm will have greater relevance, in which, in addition to parallelizing procedures, step-by-step parallelization of the ELT process is used.

***Step 1***. Create an interaction matrix between procedures and tables.

Each of the procedures executes a specific set of algorithms, while interacting with a number of tables. All existing relationships between tables must be identified. To define a set of relationships, it is required to create a mapping matrix between the tables used and the SQL statements that execute on them.

***Step 2***. Compiling Dependency Sets Between Tables.

This step is preparatory for step 3. It is required to draw dependencies between tables in the format: $T_a$ -> $T_b$ $T_c$ $T_d$. It is possible to get a set of dependency data from an interaction matrix built on step 1. The dependency is determined by the following rule:

***Rule 1***. If in the procedure $P_a$ the insert, update, or delete action is performed on the table $T_a$ and a select action for the table $T_b$, then table $T_a$ will be a dependent table on $T_b$ and this link will be recorded as follows: $T_b$ -> $T_a$.

***Step 3***. Compiling Dependency Sets Between Procedures.

Based on the set of dependencies between tables obtained at step 2 and the interaction matrices obtained at step 1 it is required to create dependency sets between procedures. This is done according to the following rule:

***Rule 2***. If in the procedure Pa the insert, update, or delete action is performed on the table $T_a$, then all the procedures in which for the table $T_a$ use select action will be procedure dependent $P_a$.

***Step 4***. Iterative optimization of ELT process by step-by-step parallelization.

After obtaining a set of dependencies between procedures, it is required to iteratively compose sets of parallel chains combined into one ELT process. This method introduces the concept of conditional dependency, which indicates the start of a procedure dependent on this type not sequentially, but after the execution of a certain set of procedures. This type of link is denoted as follows: -> $P_a$ $P_b$ $P_c$ and means that the procedure $P_c$ will be started after the procedures $P_a$ and $P_b$, does not depend on the time of further procedures

For an improved version of the previous algorithm, the principle of step-by-step parallelization of the ELT process will be used. To do this, create additional blocks for procedures that are not dependent on the execution of subsequent.

Tables relationship:

T1 -> T2, T8; T2 -> T3, T4, T6, T8; T3 -> T5, T9; T4 -> T5, T9; T5 -> T7, T8; T6 -> T8, T11; T7 -> T9, T15; T8 -> T9; T9 -> {}; T10 -> T9; T11 -> T9; T12 -> T10; T13 -> T10; T14 -> T13; T15 -> T9

Based on this data, links between procedures:

P1-> P2, P4, P6; P2-> P3, P7; P3 -> P5, P6; P4-> P6, P8; P5-> P7, P14; P6-> P7; P7-> {}; P8-> P11, P12; P9-> P11, P13; P10-> P9; P11-> P7; P12-> P7; P13-> P7; P14-> P11

Similar to the previous method, procedures that are not on the right side are selected. Such procedures are P1 and P10. They can be distributed into 2 different independent units.

Remaining chains:

P2-> P3, P7; P3 -> P5, P6; P4-> P6, P8; P5-> P7, P14; P6-> P7; P7-> {}; P8-> P11, P12; P9-> P11, P13; P11-> P7; P12-> P7; P13-> P7; P14-> P11

P2, P4 and P9 are not on the right side. They are added to blocks, based on their dependencies, and procedures emanating from them to the right side for the next step are added as well, so they will be considered controversial:

Block 1: P1 -> (P2 ?-> P3, P7) || (P4 ?-> P6, P8)

Block 2: P10 -> P9 ?-> P11, P13

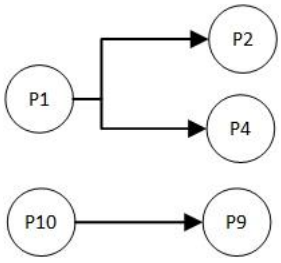


Fig. 9. Second step of iterations.

Remaining procedures:

P3 -> P5, P6; P5-> P7, P14; P6-> P7; P7-> {}; P8-> P11, P12; P11-> P7; P12-> P7; P13-> P7; P14-> P11

P3, P8, P13 are not on the right side. These procedures are put in the next iteration and remaining controversial procedures in the next step (if the procedures are duplicated, we leave only one variation) are taken:

Block 1: P1 -> ((P2 -> P3) -> P6, P5, P7) || ((P4 -> P8) -> (P11, P12))

Block 2: P10 -> P9 -> P13 ?-> P11, P7

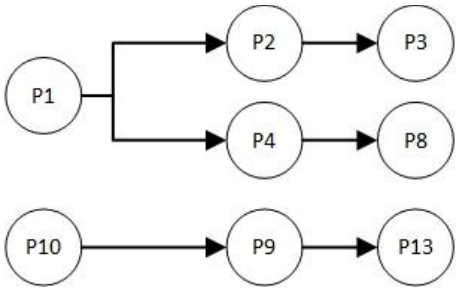


Fig. 10. Third step of iterations.

Remaining set:

P5-> P7, P14; P6-> P7; P7-> {}; P11-> P7; P12-> P7; P14-> P11

P6, P5 and P12 will be added into our chain by analogy with previous steps. Since P6 and P5 proceed from the same procedure, parallel calculation will be separated:

Block 1: P1 -> (((P2 -> P3) -> (P6 || P5 -> P14)) ?-> P7) || (((P4 -> P8) -> (P12 ?-> P7)) -> P11)

Block 2: P10 -> P9 -> P13 ?-> P11, P7

Note that in block 1 the P7 appears both in the first part of the parallel circuit and in the second. After the entire chain is completed:

Block 1: P1 -> ((((P2 -> P3) -> (P6 || P5 ?-> P14))) || (((P4 -> P8) -> (P12)) ?-> P11)) ?-> P7

Block 2: P10 -> P9 -> P13 ?-> P11, P7

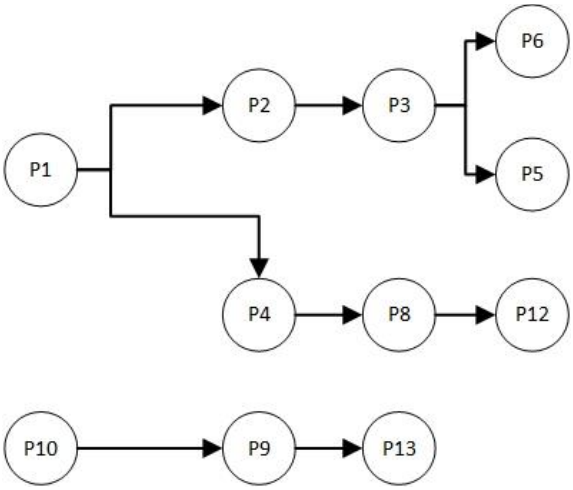


Fig. 11. Fourth step of iterations.

Remaining set:

P7-> {}; P11-> P7; P14-> P11

Adding P14 to our blocks:

Block 1: P1 -> ((((P2 -> P3) -> (P6 || P5-> P14 ?-> P11))) || (((P4 -> P8) -> (P12) ?|| P11))) ?-> P7

Block 2: P10 -> P9 -> P13 ?-> P7

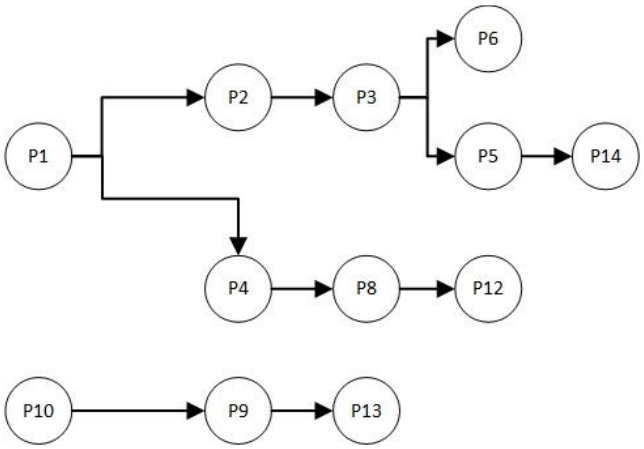


Fig. 12. Step five of iterations.

Remaining set:

P7-> {}; P11-> P7

Add P11 to our blocks:

Block 1: P1 -> ((((P2 -> P3) -> (P6 || P5-> P14 ?-> P11))) || (((P4 -> P8) -> (P12) || P11))) ?-> P7

Block 2: P10 -> P9 -> P13 ?-> P7

Note that the P11 appeared twice in the first block. In the first case, it is performed in series, and in the second in parallel. In this regard, it is taken out after executing parallel blocks and indicate conditional sequential execution as a type of connection. In the lower index of this type of relationship, the procedures after which subsequent will be recorded.

Block 1: P1 -> (((((P2 -> P3) -> (P6 || P5-> P14))) || (((P4 -> P8) -> (P12))))) ->P8P14 P11 ?-> P7

Block 2: P10 -> P9 -> P13 ?-> P7

In the remaining set, only the P7 procedure is observed:

Block 1: P1 -> (((((P2 -> P3) -> (P6 || P5-> P14))) || (((P4 -> P8) -> (P12))))) ->P8P14 P11 -> P7

Block 2: P10 -> P9 -> P13 -> P7

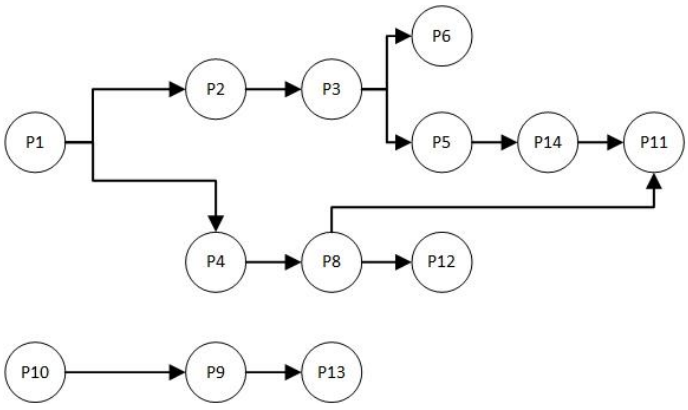


Fig. 13. Step six of iterations.

The P7 procedure appears in both blocks. In this regard, it is possible to combine two blocks into one:

(P1 -> (((((P2 -> P3) -> (P6 || P5-> P14))) || (((P4 -> P8) -> (P12))))) ->P8P14 P11) || (P10 -> P9 -> P13) -> P7

Final diagram of the ELT process:

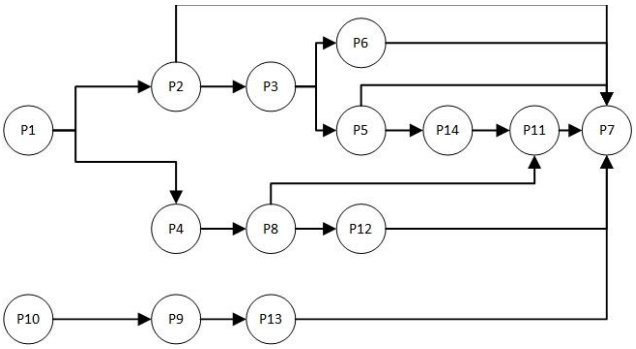


Fig. 14. Final scheme.

To calculate the time, formulas 1 and 2 are used, counting the consecutive and parallel parts of the scheme, respectively.

Conditional dependency will be considered by formula:

$$T_{agg} = B_1 + P_1 - T_{seq} \quad (3)$$

where $B_1$ – the block on which the procedure P1 conventionally depends,

$P_1$ – a procedure that depends conditionally on P1,

$T_{seq}$ – execution time of the procedure block, which follows sequentially after the execution of procedures, on which $P_1$ conditionally depends

As a result, the operating time of this chain equal to 1131 seconds is obtained.

## VIII. ELT Process Comparison

The bar diagram comparing the execution of various types of algorithms is shown on Figure 15:

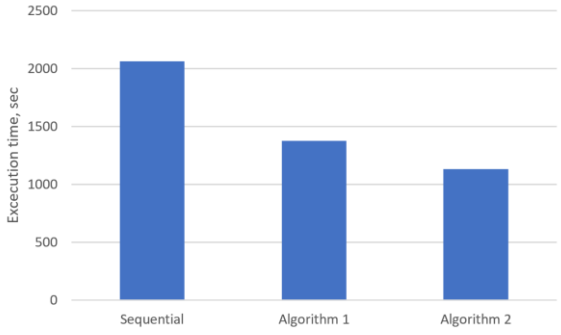


Fig. 15. Comparison of ELT algorithms.

As a result of the comparison, it was found that algorithm 1 and algorithm 2 allow for an increase in performance of more than 30 percent compared to a sequential algorithm. The best was algorithm number 2, the operating time of which was 1131 seconds.

## IX. Conclusion

This article discussed various ways to build an ELT process. Algorithms for performing PostgreSQL procedures are considered: sequential, parallelization algorithm and improved parallelization algorithm. As a result of the study, it was revealed that an improved parallelization algorithm shows the best result. It can be used in processes like trend identification [17], bionic control [18], quantum multiparty signatures [19] or AI approaches [20] where runtime is critical, with sufficient resources to perform parallel calculations. But the sequential method could be also useful when the you need to get results in a limited time because it is easy to set up.

## References


[1] E. A. Eliseeva, B. S. Goryachkin, M. V. Vinogradova, and M.V. Chernenkiy, “Estimating the execution time of search queries in NoSQL and object-relational databases”, International scientific journal "Dynamics of complex systems - XXI century": Radiotekhnika Publishing House, Moscow, 2022, No. 2, pp. 44-51.

[2] D. A. Lychagin, M. V. Vinogradova, D. S. Gudilin, and A. E. Zvonarev, “Organization of data storage in the conditions of import substitution,” Artificial Intelligence in Automated Data Management and Processing Systems, vol.1, pp.364-370, 2022. [in Russian]

[3] M. Madhikerrni and K. Främling, “Data discovery method for Extract- Transform-Load,” in 2019 IEEE 10th International Conference on Mechanical and Intelligent Manufacturing Technologies (ICMIMT), 2019, pp. 205-212, doi: 10.1109/ICMIMT.2019.8712027.

[4] Q. Hanlin, J. Xianzhen, and Z. Xianrong, “Research on Extract, Transform and Load (ETL) in Land and Resources Star Schema Data Warehouse,” in 2012 Fifth International Symposium on Computational Intelligence and Design, 2012, pp. 120-123, doi: 10.1109/ISCID.2012.38.

[5] A. Wibowo, “Problems and available solutions on the stage of Extract, Transform, and Loading in near real-time data warehousing (a literature study),” in 2015 International Seminar on Intelligent Technology and Its Applications (ISITIA), 2015, pp. 345-350, doi: 10.1109/ISITIA.2015.7220004.

[6] H. Morris et al., “Bringing Business Objects into Extract-Transform-Load (ETL) Technology,” in 2008 IEEE International Conference on e-Business Engineering, 2008, pp. 709-714, doi: 10.1109/ICEBE.2008.72.

[7] Kumari Deepika and M. Tech, “Optimisation of ETL Process Using Partitioning and Parallelization Techniques,” International Journal of Advanced Research in Computer Science, vol. 8, no. 3, 2017, doi: 10.26483/ijarcs.v8i3.3093.

[8] S.M.F. Ali and R. Wrembel, “From conceptual design to performance optimization of ETL workflows: current state of research and open problems,” The VLDB Journal, vol. 26, pp. 777–801, 2017.

[9] H. Homayouni, “Testing Extract-Transform-Load Process in Data Warehouse Systems,” 2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), 2018, pp. 158-161, doi: 10.1109/ISSREW.2018.000-6.

[10] S. K. Bansal, “Towards a Semantic Extract-Transform-Load (ETL) Framework for Big Data Integration,” in 2014 IEEE International Congress on Big Data, 2014, pp. 522-529, doi: 10.1109/BigData.Congress.2014.82.

[11] Munawar, “Extract Transform Loading (ETL) Based Data Quality for Data Warehouse Development,” in 2021 1st International Conference on Computer Science and Artificial Intelligence (ICCSAI), 2021, pp. 373-378, doi: 10.1109/ICCSAI53272.2021.9609770.

[12] A. E. Zvonarev, M. V. Vinogradova, D. S. Gudilin, and D. A. Lychagin, “Process analytics usage in the banking sector,” Artificial Intelligence in Automated Data Management and Processing Systems, vol.1, pp.47-52, 2022.

[13] Y. A. Grigoriev, “Estimation of execution time of SQL-queries to databases,” Mechanical Engineering and Computer Technologies, no. 01, 2012, http://technomag.edu.ru/doc/296143.html (accessed Feb. 7, 2031). [in Russian]

[14] Adnan, A.A. Ilham and S. Usman, “Performance analysis of extract, transform, load (ETL) in apache Hadoop atop NAS storage using ISCSI,” in 2017 4th International Conference on Computer Applications and Information Processing Technology (CAIPT), 2017, pp. 1-5, doi: 10.1109/CAIPT.2017.8320716.

[15] E. A. Eliseeva, B. S. Goryachkin, and M. V. Vinogradova, “Investigation of DBMS performance when working with cluster

databases based on ergonomic analysis," Scientific and educational journal for students and teachers STUDNET: Publishing house LLC "Electronic Science," Moscow, vol.5, no. 4, pp. 2889-2910, 2022.

[16] S. Sakulin, A. Alfimtsev, K. Kvitchenko, L. Dobkacz, Y. Kalgin, and I. Lychkov, "Network Anomalies Detection Approach Based on Weighted Voting," International Journal of Information Security and Privacy, vol. 16. pp.1-17, 2022. doi: 10.4018/IJISP.2022010105.

[17] Iu Butenko, I. Telnova, and V. Garazha, "Prospective Scientific Research Trend Identification Methods (Based on the Analysis of Gas Fuel-Related Publications)," Automatic Documentation and Mathematical Linguistics, vol. 56, pp. 11-25, 2022. doi: 10.3103/S0005105522010034.

[18] A. Briko, V. Kapravchuk, A. Kobelev, A. Hammoud, S. Leonhardt, C. Ngo, Y. Gulyaev, and S. Shchukin, "A Way of Bionic Control Based on EI, EMG, and FMG Signals," Sensors, vol. 22, no. 1, p. 152, 2021, doi: 10.3390/s22010152.

[19] E. Kiktenko, A. Zelenetsky, and A. Fedorov, "Practical quantum multiparty signatures using quantum key distribution networks," Phys. Rev. A, vol. 105, p. 012408, 2021.

[20] A. Masalimova, M. Khvatova, L. Chikileva, E. Zvyagintseva, V. Stepanova, and M. Melnik, "distance learning in higher education during Covid-19," Frontiers in Education, vol. 7, 2022, doi:10.3389/feduc.2022.822958.