Deployment approaches in distributed complex event processing

Arsenij Zorin
Southwest State University
Kursk, Russia
zorinarsenij@bk.ru

Irina Chernetskaya
Southwest State University
Kursk, Russia
white731@yandex.ru

***Abstract*—Big Data technologies have traditionally focused on processing human-generated data, while neglecting the vast amounts of data generated by Machine-to-Machine (M2M) interactions and Internet-of-Things (IoT) platforms. These interactions generate real-time data streams that are highly structured, often in the form of a series of event occurrences. In this paper, we aim to provide a comprehensive overview of the main research issues in Complex Event Processing (CEP) techniques, with a special focus on optimizing the distribution of event handlers between working nodes. We introduce and compare different deployment strategies for CEP event handlers. These strategies define how the event handlers are distributed over different working nodes. In this paper we consider the distributed approach, because it ensures, that the event handlers are scalable, fault-tolerant, and can handle large volumes of data.**

***Index Terms*—complex event processing, distributed processing, event based systems**

I. Introduction

Several complex systems operate by observing a set of primitive events that happen in the external environment, interpreting and combining them to identify higher level composite events, and finally sending notifications about these events to the components in charge of reacting to them, thus determining the overall system's behavior. This means that the systems are able to perform complex tasks by breaking them down into simpler, more manageable events. In order to achieve this, the systems use a general architecture that includes sources and sinks at the peripherals of the system. These sources observe primitive events and report them, while the sinks receive composite event notifications and react to them.

At the center of the system is the complex event processing (CEP) subsystem, which is responsible for processing and routing events from sources to interested sinks. It operates by interpreting a set of event definition rules, which describe how composite events are defined from primitive ones [1, 2]. The CEP subsystem is crucial to the operation of the system, as it is responsible for ensuring that the right events get to the right places.

Event-based applications usually involve a large number of sources and sinks, possibly dispersed over a wide number of working nodes [3, 4, 5]. This means that the CEP subsystem can be internally built around several, distributed working nodes, connected together to form an overlay network, and cooperating to provide the processing and routing service [6]. This allows the system to process and route events more efficiently, as it can distribute the workload across multiple working nodes.

This paper introduces and compares different deployment approaches for CEP, which are designed to optimize the performance of the system. A deployment approach defines how the event handlers are distributed over working nodes. The first aspect is often called operator placement, and it involves finding the best mapping of the event handlers defined in rules on available working nodes [7]. Operator placement may pursue different goals, such as reducing the latency required to deliver notifications to interested parties, or minimizing the usage of network resources. In the last few years, different solutions have been proposed for operator placement. However, the problem is known to be extremely complex to solve, even for small instances with a reduced number of workers and rules. Accordingly, existing approaches are often based on approximated optimization algorithms or heuristics, and they usually rely on a centralized decider, which collects all the relevant information about the network status and locally computes a solution to the problem.

The novelty of this work is the study of the applied use of scaling approaches in systems for processing complex events in real time. The solutions presented in this paper are explicitly tailored to large scale distributed scenarios. They try to take into account the topology of the processing network as well as the location of event sources and their generation rates [8].

II. approaches

A. Uniform distribution of handlers between working nodes

This approach for distributing handlers is based on an even distribution of handlers among all the working nodes. The implementation of this approach is simple and requires a few steps. Firstly, the handler distribution storage must be expanded to include information about the number of running handlers on each of the working nodes. The data schema in DBML format might look like this:

```
Table handlers {
  id integer [primary key]
  w_node_id integer
  other_data data
}
```

```
Table working_nodes {
  id integer [primary key]
  other_data data
}

Ref: working_nodes.id > handlers.w_node_id
```

The volume of the information storage depends on the number of working nodes and handlers, but does not depend on the number of events occurring in the system. Therefore, the memory cost for storing the information can be estimated in $O(W + H)$, where W is the number of working nodes, and H is the number of handlers.

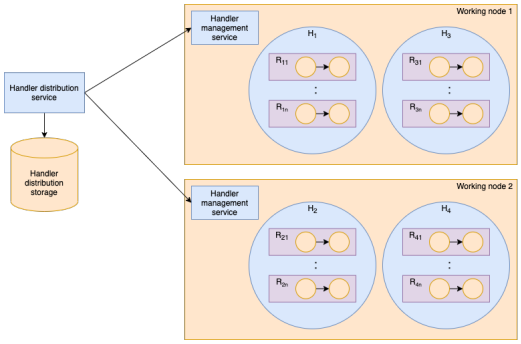

Fig. 1. Uniform distribution of handlers between working nodes.

Once this information is available, the handler distribution service can be used to control the even launch of handlers across all working nodes. Fig. 1 illustrates this approach with the uniform distribution of four handlers between two working nodes. The handler distribution storage is used to store information about the handlers that are running on specific working nodes and their numbers. If there is a change in the number of handlers, the handler distribution service will redistribute them. When a new handler is added, the handler distribution service identifies the working node with the fewest running handlers and deploys the new handler to that node. Conversely, when a handler is removed, the handler distribution service removes information about the handler from the handler distribution storage and sends a handler shutdown command to the handler management service. However, removing handlers may cause an imbalance in the number of handlers on each working node.

To solve this issue, the handler distribution service periodically balances the number of handlers on each working node. The service first determines the maximum number of handlers allowed on each working node using the following formula:

$$N = \left\lceil \frac{H}{W} \right\rceil \tag{1}$$

In (1) H is the number of event handlers and W – the number of working nodes. It then sequentially traverses the sorted list of working nodes, and if the number of running handlers on the working node is more than the maximum number allowed, the service searches for working nodes with a number of handlers less than the maximum allowed. The excess handlers from the current working node are transferred to the new working nodes. The handlers redistribution algorithm will look like this:

Algorithm 1 Function RedistributeHandlers(W,H)

$w_{start} \leftarrow 0$
$w_{end} \leftarrow len(W) - 1$
$n \leftarrow len(H)/len(W)$
for $w_{start} \leq w_{end}$ **do**
  if $W[w_{start}].number\_of\_handlers < n$ **then**
    for $w_{start} \leq w_{end}$ **do**
      if $W[w_{end}].number\_of\_handlers > n$ **then**
        $Redistribute(W[w_{start}], W[w_{end}])$
        if $W[w_{start}].number\_of\_handlers \geq n$ **then**
          $break$
        end if
      end if
      $w_{end} \leftarrow w_{end} - 1$
    end for
  end if
  $w_{start} \leftarrow w_{start} + 1$
end for

The asymptotic complexity of the algorithm in such an implementation is equal to $O(max(W, H))$. Although this approach is easy to implement and allows for horizontal scaling of handlers, it has some inherent disadvantages. For instance, it does not take into account the internal complexity of each handler or possible differences in the number of resources on the working node. Each handler may contain a different number of rules, and the frequency of rule triggering may vary. Additionally, working nodes may have differing amounts of resources, which can lead to low efficiency in the distribution of handlers across working nodes.

B. Distribution of handlers based on the number of rules

This approach shares similarities with the previous one, but there is a key difference in how the handlers are distributed. Instead of relying on a simple criterion, such as the number of active handlers, this approach takes into account the number of handlers running on each working node. To accomplish this, the handler distribution storage is expanded to include information about the number of rules in each handler. The extended data schema in DBML format for that approach might look like this:

```
Table handlers {
  id integer [primary key]
  number_of_rules integer
  w_node_id integer
  other_data data
}

Table working_nodes {
  id integer [primary key]
```

```
  other_data data
}

Ref: working_nodes.id > handlers.w_node_id
```

This allows for a more nuanced approach to balancing the workload between working nodes, which is illustrated on fig. 2.

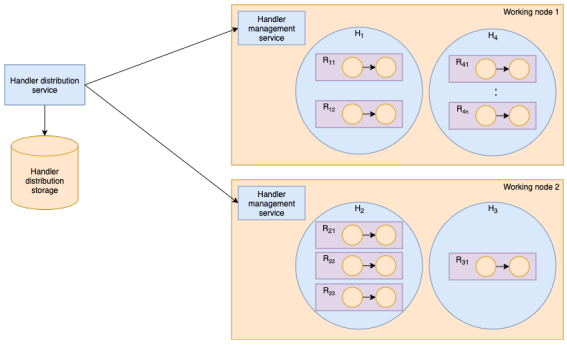

Fig. 2. Distribution of handlers based on the number of rules

The volume of the information storage depends on the number of working nodes and handlers as for the previous approach. Therefore, the memory cost for storing the information can be estimated in $O(W + H)$. The redistribution algorithm requires an analysis of the number of rules executed on the working node, instead of calculating the number of handlers. The complexity of the algorithm corresponds to the complexity of the previous algorithm and is equal to $O(max(W, H))$.

One of the main advantages of this approach is that the handler distribution service can monitor the total number of handler rules running on each working node. Like the previous approach, the handler distribution service performs balancing at fixed intervals. However, the key difference is the inclusion of additional information about the number of rules, which allows for a more complex balancing algorithm to be used. By evenly distributing handlers, this approach minimizes the number of rules executed on each working node, which can lead to more efficient processing. However, it's important to note that this approach still does not take into account the frequency of rule firing or the different amounts of available resources on working nodes, which could impact overall performance. Therefore, it may be necessary to explore additional strategies for optimizing the workload distribution in the future.

C. Distribution of handlers based on the configuration of the required resources

This approach involves a preliminary configuration of the necessary resources for each handler. The system administrator adds information about the resources that are needed for each handler and also adds information about the resources available on each working node. With the help of this information, the distribution of handlers between working nodes takes place. The distribution process ensures that the resources of working nodes are utilized as much as possible. Before launching a network of handlers, the configuration of the resources required by each handler and the resources available on each working node is performed. The configurable resources can be the number of CPU cores and the size of RAM. In addition to being able to configure resources, this approach also allows for consideration of the frequency of execution of the rules by each handler. This frequency data could be used to optimize the distribution of handlers.

The volume of the information storage depends on the number of working nodes and handlers as for the previous approach. Therefore, the memory cost for storing the information can be estimated in $O(W + H)$. The extended data schema in DBML format for that approach might look like this:

```
Table handlers {
  id integer [primary key]
  cpu_required integer
  memory_required integer
  w_node_id integer
  other_data data
}

Table working_nodes {
  id integer [primary key]
  cpu integer
  memory integer
  other_data data
}

Ref: working_nodes.id > handlers.w_node_id
```

The task of efficiently placing handlers in this approach is an NP challenge. Therefore, a resource allocation approach from kubernetes can be used to provide a trade-off between speed and efficiency [9]. In this case, the algorithm is reduced to calculating the estimate of the deployment of the handler on each of the working nodes [10]. The algorithmic complexity of this algorithm is $O(W * H)$.

The scheme of this approach is shown in fig. 3.

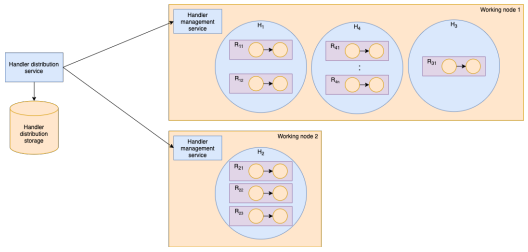

Fig. 3. Distribution of handlers based on the configuration of the required resources

However, one disadvantage of this approach is the need for manual configuration of allocated resources, which can be time-consuming. Another disadvantage is that this approach does not take into account the dynamic nature of resource

availability, which could lead to suboptimal resource utilization. To address these limitations, future research could explore the use of machine learning algorithms to automate the allocation of resources and dynamically adjust to changes in resource availability.

D. Distribution of handlers based on statistics collected during operation

All previous diagrams are based on information obtained from starting the entire system and creating new handlers. However, it is not always possible to determine how many resources to allocate to a handler and on which working node it is most efficient to place them. This problem is due to the fact that at the time the handlers are launched, there is no information about the frequency of the rule's operation. It is important to consider the frequency of rule execution when allocating resources because it can affect the efficiency of the handler. A handler may contain a large number of rules, but these rules are fired quite rarely [11]. In contrast, a handler may contain only one rule, but fire on most events. These scenarios can lead to resource waste or inefficient allocation. One way to solve this problem is to collect analytics from handlers while the system is running. Collecting statistics on the execution time and frequency of rules can help in balancing handlers with infrequently executed rules on less productive working nodes and those with the longest rule execution time and high execution frequency on high-performance working nodes. To collect statistics, it is most efficient to run the statistics storage locally on each working node. This will ensure the shortest time to send statistics from the handler to the statistics storage. Each handler sends all necessary statistics to the local statistics storage on the working node. The handler distribution service collects handler statistics from each working node through the handler management service during balancing. After that, the service aggregates the collected statistics and, based on the results, redistributes highly loaded processors to the most high-performance working nodes. This ensures that the system is balanced and optimized for efficient execution. The extended data schema in DBML format for that approach might look like this:

```
Table rules {
    id integer [primary key]
    processing_time_q95 integer
    number_of_activations integer
    h_id integer
}

Table handlers {
  id integer [primary key]
  w_node_id integer
  other_data data
}

Table working_nodes {
  id integer [primary key]
  cpu integer
  memory integer
  other_data data
}

Ref: working_nodes.id > handlers.w_node_id
Ref: handlers.id > rules.h_id
```

The volume of the information storage depends on the number of working nodes, handlers and rules. Therefore, the memory cost for storing the information can be estimated in $O(W+H+R)$, where W is the number of working nodes, H is the number of handlers and R is the number of rules. Also this approach uses local storage for rule execution statistic. This collected statistic can be collapsed, so the space used does not exceed $O(R)$, since all statistics are duplicated in the handler distribution storage.

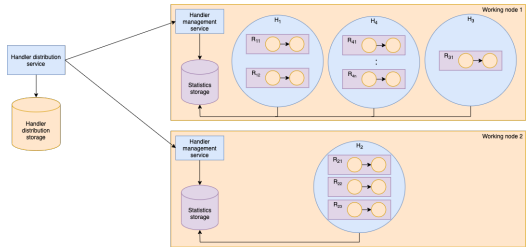

Fig. 4. Distribution of handlers based on statistics collected during operation

On fig. 4, we can see the distribution of handlers based on the statistics collected during the work. The diagram shows that each working node has local statistics storage. The handler distribution service, at the time of balancing, collects and aggregates data from local statistics storages and creates it. So, as shown in fig. 4, the handler distribution service receives information about the 95th percentile of the rule execution time and the number of rule firings. Based on the aggregated statistics, the handler distribution service performs balancing and places the most loaded H_2 handler on a separate working node 2. This algorithm also reduces to solving the bin packing problem, like the previous one, and has a similar complexity - $O(W * H)$

In conclusion, collecting analytics can help in efficient resource allocation and balancing of handlers, leading to a more optimized system. By running the statistics storage locally on each worker node, the system can ensure the shortest time to send statistics from the handler to the statistics storage.

III. Comparison of approaches

Let's make a comparative analysis of the described schemes for working with events according to the following criteria [12]:

- Support for working with working nodes with different amounts of resources;
- Level of support for accounting for the frequency of operation of handler rules;

- The need to develop additional services and repositories with information storages;
- The complexity of the algorithm for redistributing handlers between working nodes.

Consider the rating scale for each criterion. The criterion for supporting work with working nodes with different amounts of resources can be evaluated on the following scale:

- Present - 1;
- Absent - 0.

The criteria for the level of support for accounting for the frequency of triggering of handler rules can be assessed on a scale:

- Dynamic support - 1;
- Static support - 0.5;
- Absent - 0.

Dynamic support implies the ability of the system to independently collect statistics on the frequency of rule triggering and, based on the collected data, balance handlers. Static support allows configuration of the frequency of rule triggering at the system startup stage. This approach does not allow efficient utilization of resources in the case of a changing frequency of rule firings over time.

The criteria for the need to develop additional services and repositories can be estimated based on the assessment of overhead costs for information storage. Thus, the criterion can be assessed on the following scale:

- Development of additional services and repositories is not required, no overhead - 1;
- Requires the development of information storage, the volume of which does not depend on the number of rules specified - 0.5;
- Requires the development of information storage, the volume of which depends on the number of rules or a value of a higher order - 0.

The criteria for the complexity of the algorithm for redistributing handlers between working nodes can be estimated using the following scale:

- Algorithm complexity not exceeding O(max(W,H)) - 1;
- Algorithm complexity not exceeding O(W * H) - 0.5;
- Algorithm has quadratic complexity and higher - 0.

Criteria 1 and 2 are the most important as they affect the efficiency of resource utilization at working nodes [13]. Therefore, the weight of criteria 1 and 2 is 0.3, and the weight of criterion 3 and 4 is 0.2. The weighted sum method shows (Table 1) that the approach of distributing handlers based on run-time statistics is more appropriate.

It allows working with working nodes that have different amounts of resources and provides a redistribution of handlers between working nodes, taking into account the actual frequency of rule firing. This approach also has disadvantages in the form of the need to create additional local storage of statistics and implement the aggregation of the collected statistics.

TABLE I
COMPARISON BY WEIGHTED SUM METHOD

	Approaches			
Criteria	***A***	***B***	***C***	***D***
C_1	0	0	1	1
C_2	0	0	0.5	1
C_3	0.5	0.5	0.5	0
C_4	1	1	0.5	0.5
Weighted sum	0.35	0.35	0.65	0.7

IV. CONCLUSION AND FUTURE WORK

Having thoroughly reviewed the state-of-the-art approaches that focus on efficient event handler distribution and can be applied in CEP systems. We have come to the conclusion that the approach using statistics collected during the operation of the system to redistribute handlers between working nodes is the most suitable approach for modern systems. This approach utilizes not only the static configuration of the distribution strategy at the stage of system startup but also dynamic redistribution based on statistics collected during the operation of the system. This can improve the efficiency of resource utilization in the system. Therefore, we recommend that future research focus on the study of hybrid approaches to managing the distribution of handlers between working nodes, where both static configuration and dynamic redistribution can be used to maximize system efficiency.

In addition to this, we suggest that it would be beneficial to select the optimal set of metrics that can effectively redistribute event handlers. Further research in this area may lead to the identification of the most relevant metrics.

Although we have considered centralized approaches to managing the distribution of event handlers in this work. There are also decentralized approaches that provide a higher level of fault tolerance and have the potential to scale efficiently [14,15]. Therefore, we suggest that future work may explore these decentralized approaches as well. By investigating both centralized and decentralized approaches, we can gain a better understanding of the advantages and disadvantages of each and ultimately identify the best approach for a given system.

REFERENCES

[1] A. Paschke, and A. Kozlenkov, “Rule-Based Event Processing and Reaction Rules: Lecture Notes in Computer Science,“, pp. 53-66, 2009.

[2] G. Cugola, and A. Margara, “Deployment strategies for distributed complex event processing: Computing,“, vol. 95, no. 2, pp. 129-156, 2012.

[3] M. Fardbastani, and M. Sharifi, “Scalable complex event processing using adaptive load balancing: Journal of Systems and Software,“ v. 149, pp. 305-317, 2019.

[4] A. Sun, Z. Zhong, H. Jeong, and Q. Yang, “Building complex event processing capability for intelligent environmental monitoring: Environmental Modelling and Software,“ v. 116, pp. 1-6, 2019.

[5] D. Loreti, F. Chesani, P. Mello, L. Roffia, F. Antoniazzi, T. Cinotti, G. Paolini, D. Masotti, and A. Costanzo, “Complex reactive event processing for assisted living: The Habitat project case study: Expert Systems with Applications,“ v. 126, pp. 200-217, 2019.

[6] E. Brazález, H. Macià, G. Díaz, M. Baeza Romero, E. Valero, and V. Valero, “FUME: An air quality decision support system for cities based on CEP technology and fuzzy logic: Applied Soft Computing,“ v. 129, pp. 109536, 2022

[7] A. Paschke, and A. Kozlenkov, “Rule-Based Event Processing and Reaction Rules: Lecture Notes in Computer Science,“ pp. 53-66, 2009.
[8] A. Alakari, K. F. Li, and F. Gebali, “A situation refinement model for complex event processing,“ Knowledge-Based Systems [online] 198, 105881, 2020.
[9] K. Hightower, B. Burns, and J. Beda, “Kubernetes: Up and Running: Dive into the Future of Infrastructure,“ O’Reilly Media, 2017.
[10] M. Luksa, “Kubernetes in Action,“ Hanser Fachbuchverlag, 2018, ISBN 9783446455108.
[11] D. Wang, M. Zhou, S. Ali, P. Zhou, Y. Liu, and X. Wang, “A Novel Complex Event Processing Engine for Intelligent Data Analysis in Integrated Information Systems: International Journal of Distributed Sensor Networks,“ vol. 12, no. 3, pp. 6741401, 2016.
[12] A. Alakari, K. F. Li, and F. Gebali, “A situation refinement model for complex event processing,“ Knowledge-Based Systems [online] 198, 105881, 2020.
[13] A. Margara, and G. Cugola, “High-Performance Publish-Subscribe Matching Using Parallel Hardware: IEEE Transactions on Parallel and Distributed Systems,“ vol. 25, no. 1, pp. 126-135, 2014.
[14] G. Cugola, and A. Margara, “Complex event processing with T-REX: Journal of Systems and Software,“ vol. 85, no. 8, pp. 1709-1728, 2012.
[15] S. Jayasekara, S. Kannangara, T. Dahanayakage, I. Ranawaka, S. Perera, and V. Nanayakkara, “Wihidum: Distributed complex event processing: Journal of Parallel and Distributed Computing,“ vol. 79-80, pp. 42-51, 2015.