NATIONAL RESEARCH CENTRE
«KURCHATOV INSTITUTE»
Institute for High Energy Physics
of the National Research Centre
«Kurchatov Institute»

Preprint 2020–5

M.V. Golosova[1], V.A. Aulov[1], V.V. Kotliar[2]

Architecture development of the management system for topologies of metadata integration processes

[1]NRC "Kurchatov Institute", 1, Akademika Kurchatova sq., Moscow, 123182, Russia
[2]NRC "Kurchatov Institute" – IHEP , 1, Nauki sq, Protvino, Moscow region, 142281, Russia

Protvino 2020

UDK 539.1.074.3 M-24

Abstract

Golosova M.V., Aulov V.A., Kotliar V.V. Architecture development of the management system for topologies of metadata integration processes: NRC «Kurchatov Institute » – IHEP Preprint 2020–5. – Protvino, 2020. – p. 17, figs. 5, refs.: 7.

Information integration is an ever-lasting problem in IT. It can be done on demand – manually or in an automated way – or as a background service that prepares integrated information for further usage. To provide integration as a service, two types of questions should be addressed: case-specific (which data should be integrated, how to get them and how to prepare) and case-independent (how to organize process, make it reliable and scalable, monitor its state). This paper describes a conceptual architecture of the Metadata Integration Topology Management System (MInTMS), designed to simplify the development and management of integration processes in a case-independent manner.

Аннотация

Голосова М.В., Аулов В.А., Котляр В.В. Разработка архитектуры системы управления топологиями процессов интеграции метаданных: Препринт НИЦ «Курчатовский институт» – ИФВЭ 2020–5. – Протвино, 2020. – 17 с., 5 рис., библиогр.: 7.

Проблема интеграции информации в области информационных технологий существует уже довольно давно. Интеграция может осуществляться по требованию пользователя – вручную или автоматизировано – или как сервис, который в фоновом режиме подготавливает интегрированное представление для дальнейшего использования. Для обеспечения интеграции как сервиса, должны быть решены задачи двух типов: специфичные для каждого конкретного случая (какие данные необходимо интегрировать, как их получить и как они должны быть подготовлены) и общие для всех случаев (как организовать процесс интеграции, как сделать его надёжным и масштабируемым, как контролировать его состояние). Эта статья описывает концептуальную архитектуру системы управления топологиями интеграции метаданных, спроектированной как средство для упрощения разработки и управления процессами интеграции в общем виде.

1. Introduction

Information integration is an inevitable question in the world of IT. Huge volumes of information are already stored in and indexed by numerous different systems, ranging from a local file with a few lines of text and to the Google index, and can be obtained via different UIs and APIs. Today it is not infrequent that the one looking for some information has to combine it from multiple different sources to get the desired result.

In case of a one-off search, it is fine to query all sources manually – and the more so because the results of one query may determine the next step in the chain of queries. However, there are many use-cases when such a search scenario becomes a regular routine and its automation is a natural way to optimize the workflow, providing users with automated integration on demand. Yet there also are some use-cases with too much information to be integrated (so that automated search scenario cannot be executed fast enough to promptly provide users with desired results, e.g. due to the response time of the original information resources) or too many parametric options that cause changes in the scenario (which significantly complicate its automation and optimization). For example, it can be monitoring of intensive operations in large projects that handles huge volumes of different information collected and stored separately (even if it is just a different tables of same relational database) – or business analytics that requires fast aggregation of all available information on different levels of abstraction and by multiple different parameters, depending on the current focus of interest. In cases like these execution of the integration on demand may not be enough for effective information use.

When the integration on demand is not applicable, it can be executed in the background: collecting and linking together all the information of interest in advance and storing it in the most optimal way for further usage. In this case, compared to the integration on demand, there are many new issues to be addressed by developers, one of which, of course, is where and how the integrated information should be stored for further usage. It is to be decided for each situation individually, for the answer heavily depends on the character of the information, its volume and how it is supposed to be used.

Other issues, however, are more technical and do not depend on the specifics of a given use-case. For automatic background integration, the following questions should be decided:

- how to get integrated not only the requested right now, but all the information;

- how to keep the integrated information up to date and consistent with the original sources of information;
- how to implement integration scenarios to make them fault tolerant, prevent data loss, avoid excessive resource usage, etc.;
- how to scale implemented scenarios (e.g. to reduce time lag between integrated information and original sources);
- how to monitor integration scenarios execution state;
- how to minimize integration scenarios management time expenditure.

Metadata Integration Topology Management System (MInTMS) is aimed to simplify these technical questions by providing a ready-to-use set of tools that would allow users to set up and manage automated information integration for any use-case (after solving the case-specific questions). This paper provides conceptual design for the system, discusses possible technological solutions for selected components and describes system prototype, developed as a part of the DKB project [1].

2. MInTMS: conceptual design

MInTMS is designed to operate with information in the form of metadata: textual (or numeric) description of information object and its properties, where object is seen as an object itself (of a material or digital world) or a process. The system supports all possible types of original sources of information (databases, documents, spreadsheets and other types of files) – as long as there's a way to automate the metadata extraction from them. Likewise, the final storage (dedicated storages for the integrated information) can also be of any type – as long as the data load process can be automated.

Although the system is designed to simplify the development of the integration process, it does not provide users with a high-level language to describe the integration process with, or a predefined set of operations to be combined into a desired workflow. Instead, it provides a framework to combine the user's implementation of the desired data operations into a seamless, reliable, easily scalable and manageable integration process.

This section describes main components and concepts of the MInTMS architecture and their functionality without references to the underlying technologies. More detailed discussion on the components' implementation can be found in section 3.

2.1. Overview

High-level management operations are executed by the Main Server – a component of MInTMS providing users with an interface that allows operating with integration scenarios as individual logical items. After a new scenario is defined, it can be enabled or disabled to start/stop the automatic integration, reset to start integration anew or from specified point, modified and purged to remove integrated data – or updated to apply changes to the already integrated data without full data removal from the final storages.

To execute the integration scenario in a distributed mode, Main Server relies on the Resource Manager. The RM distributes multiple instances of integration scenario Builders, initialized by the Main Server, over available resources. These Builders are responsible for instantiation of specific parts of the scenario according to the scenario configuration, instructions from the Main Server and load balancing requirements. Each part of the scenario, in its turn, is implemented within a scenario construction framework, which takes care of execution of the user defined operations in a proper order and manages communication between the independently operating parts of the whole scenario.

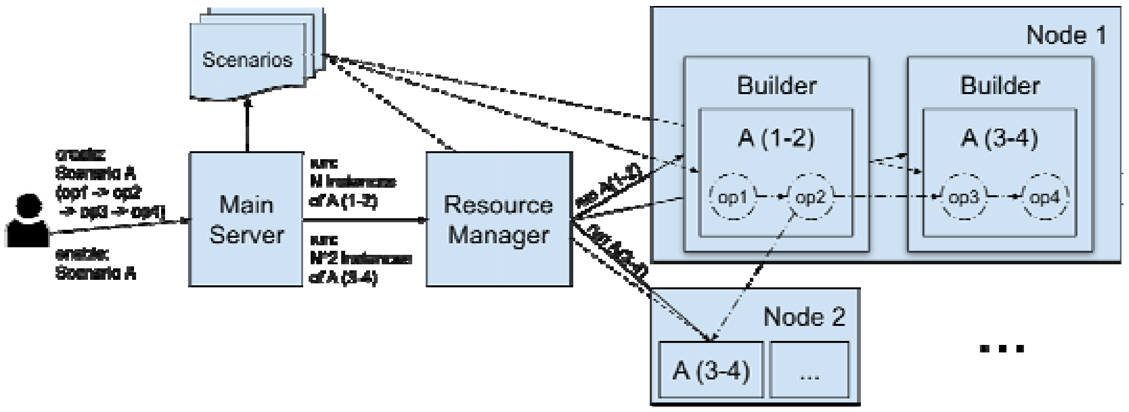

Figure 1. MInTMS main components.

2.2. Integration scenario construction framework

The integration scenario itself may be presented as data flow from sources to sinks (final storages) shaped by the use-case defined ETL process (process of information extraction, transformation and load) (Figure 2). As long as the MInTMS operates with metadata, data flow is considered to contain structured information that can be represented as a series of independent information units – messages – each of which contains a description of a single object or object's property. Although the sources may generate a mixed flow that contains messages in different formats and/or related to objects of different types, it would complicate the logic of the rest of the ETL process – for each operation may depend on the type of messages and have different implementation for each type; and even the whole set of required operations may vary. To avoid this complication, the data flow is assumed to be uniform, unless before the stage that splits data flow into a number of flows according to the message types. For different types of messages it is suggested to define different integration scenarios or add a switch point after which the scenario branches into independent sub-scenarios.

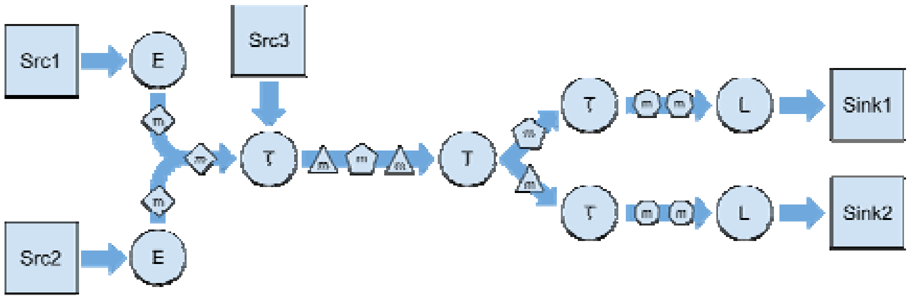

Figure 2. ETL process and data flow from sources to final storages. Differently shaped messages illustrate messages of different types and/or formats.

The shape of the data flow, switch points, scenarios of data-related operations (extraction from original sources, transformation on the way to the sinks and, finally, load to the final storages), as well as execution rules for these scenarios – all this together makes up a topology of the integration scenario.

Integration scenario is a main operational unit of the Main Server. It is unique for every use-case, and its definition in terms of MInTMS is nothing else but a formalized interpretation of the user's understanding of what should be done to bring together information from original sources and write it in a desired form into the final storages. Construction of this interpretation (given that the analysis of the use-case – what is the information of interest, from which sources it should be taken, what is to be the "product" of integration and how it should be stored – is already done) takes the following steps:

- detect the main source (or sources) – source allowing extraction of new information (added or updated after the last extraction operation); other sources will be used as secondary ones, providing some additional information to that extracted from the main source;

- split the whole process into (semi-)independent logical stages:
    - one extraction (E-) stage for each main source;
    - one load (L-) stage for each final storage;
    - one transformation (T-) stage for each additional source;
    - additional transformation stages for:
        - adjusting information representation to the final storage schema;
        - introduction of surrogate keys and derived values calculation;
        - splitting and joining data flows;
        - format conversion;
- construct an acyclic digraph, vertices and edges of which represent, respectively, the defined stages and flows of data between them – this is a logical scheme of the integration scenario, defining what operations and in what order should be performed;
- implement stages not having standard implementation (which may be presented for data flows operations with no or low dependency on the actual content of the flows, like flows joining and splitting) and not implemented earlier in other use-cases;
- provide graph vertices with instructions for related stages execution (starting instructions, schedule for E-stages, parallelization parameters, etc.).

Multiple main sources may be required in case when new information about same or related objects is generated simultaneously by more than one source: e.g. when user interacts with more than one system simultaneously, his or her actions will be logged by those systems independently – but for analytics purpose it may be useful to join these actions by known patterns, which may be done during the integration process. It may be organized as a scenario with a single main source as well, in which case the second system's logs will be used as a secondary source – but it will also mean that updates from the secondary sources won't reach the final storage unless there are updates in the primary one.

Another way to implement the integration scenario with multiple main sources is to develop a set of similar scenarios, each of which has single main source and takes others as secondary. Each scenario in this set will produce complete representation of objects marked as new or updated in its main source; some objects may be processed more than once (if updated in more than one source), but independence of scenarios in the set means that update procedures can be distributed in time to balance resource usage.

To avoid re-processing of the same object multiple times, all main sources may be queried at the same time; produced records then will be merged at the first T-stage by a specific key, after which a set of T-stages will query main sources as secondary ones for those messages that do not contain information from a specific source. In this case resource usage will be more localized in time, but in a single run of the scenario none of the ETL-operations will be executed more than once for the same object.

Constructed graph contains all the information required to build the whole topology of the scenario, start metadata integration and fill the final storages with data prepared for further usage. In sum, the description of the scenario, provided by the user to the Main Server, contains list of topology nodes (logical operations to be performed on data), types of nodes (in terms of ETL-process: E-, T-, or L-), links between them (order in which operations should be applied to the data flow) and information on how to execute logical operations represented by each node (implementation and starting instructions).

Depending on the use-case the combination of nodes and corresponding operations implementation varies – while the data flow management operations (messages transfer between nodes, data extraction triggering, etc.) are more universal and can be reused in any other scenario. These universal operations are turned into some kind of a topology construction framework, where:

- *topology* is the instance of the integration scenario or its part consisting of a single node or multiple connected nodes;
- *topology nodes* (of three types: E-, T-, L-) take care of applying operations provided by the user to data in the flow: E-node – according to the schedule, T- and L-nodes – on demand (when there are some data in the input data flow);
- *data channels* are connections between nodes and topologies; they link one node's output to the one or multiple "downstream" nodes' input (where the up- and downstream nodes may or may not belong to the same topology) and guarantee that every message will be delivered to each node exactly once (meaning

that no data will be lost, yet at the same time no data will be processed more than once). In some use-cases, when resulting records in the final storages do not depend on the number of times the original message was re-processed (all the transformations are completely independent, or stateless), the requirement of exactly-once message delivery can be softened to “at least once”. However even in these cases, especially if data flow is very intense, avoiding unnecessary operations will help to reduce processing time, making integration process more efficient;

- *Integration State Store* keeps current state (information about last data that were processed) of the E-nodes.

As for the implementation of the use-case-specific operations, provided by the user – they take the shape of pluggable into this framework modules called workers. Workers objective is to perform user defined actions on the data in the flow on demand. Depending on the type of the corresponding node, the action is:

- E: extract data from main source, wrap them into messages and output the result (to be passed for further processing);
- T: process (transform) the input message and output the result in form of new messages;
- L: upload the input messages content to the final storage.

To make the MInTMS suitable for a wide variety of users – developers and development teams with different backgrounds and routine preferences – the way workers should be implemented is not restricted to a specific programming language or a set of languages. The only requirement is that workers must be provided as executables pluggable to the topology nodes of the corresponding types – in other words, to be able to communicate with topology nodes (also referred to as node supervisors).

Since all messages are considered to be independent, T- and L-workers need to know only how to process a single message. However, in order to avoid additional expenses due to the worker restarting for each message in the data flow, they also should be able to operate with the flow of messages: start processing when there are new data at the input (or when triggered) and go into standby mode when there’s nothing to do. Figure 3 illustrates how data flow goes through the topology nodes in terms of supervisors and workers: E-node supervisor triggers worker according to the ETL-process schedule and waits for messages it would generate; as soon as a message is received, supervisor sends it to the output channel and, after the last message, goes into a standby mode until it's time to query the source for new data. Messages produced by the E-node are delivered to the connected T-node supervisor, which sends them to the associated worker for processing and gets back transformed messages, which, in turn, are sent to the output channel of the T-node. After the T-node (or a chain of T-nodes) the data finally get to the L-node; it sends messages to the worker just like any of T-nodes, but does not expect data on the worker's output (so technically, L-node is a T-node with empty output).

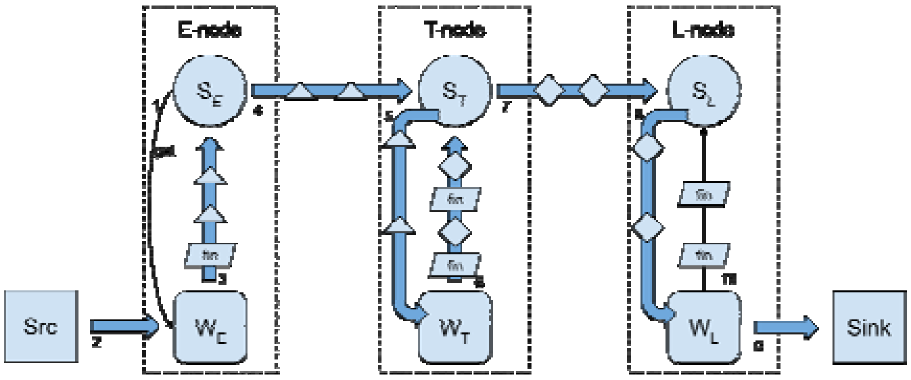

Figure 3. Data flow in topology nodes: supervisor/worker interaction.

By design, data channels between supervisors can guarantee message delivery from node to node; the nodes, in turn, can use this mechanism to guarantee message processing: messages are not marked as "received" unless worker confirms successful processing ("fin" marker in Figure 3). It makes requirements to the communication channels between supervisor and worker very soft: even if some data were lost or damaged during transfer, it means only that supervisor needs to send the same message again; and even if the operation was interrupted, on the restart supervisor will skip all the properly processed messages and start over from the first unprocessed one. Soft requirements allow usage of very simple data transfer protocols, making workers implementation easier. It is very important, since it means that a worker can be implemented from scratch in any programming language without special libraries providing MInTMS functionality.

3. Apache Kafka based architecture

Two components of the MInTMS, data transfer channels between nodes and Integration State Store, carry out very common tasks: to store and transfer data. In different situations these tasks can be addressed in multiple different ways based on different communication protocols and approaches to data management, and many of these solutions are already implemented.

For connections between integration topology nodes in MInTMS authors suggest to utilize a message-oriented middleware (MoM), specifically the distributed streaming platform Apache Kafka [2]. MoMs are usually used when small amounts of data need to be exchanged frequently [3], which is fair for data exchange between nodes of the metadata processing topology: object metadata usually are light-weight (compared to the object itself), but within the topology each message with these metadata can be transferred between nodes multiple times – from source to sinks, through all transformation stages. A signature feature of the Apache Kafka for which it was chosen among others is that along with reliable message delivery from node to node it also provides a set of tools for stream processing (custom transformation of messages), assembled in the Kafka Streams library – which can be used as a basis for the Builder of the most branchy, "transformation" part of the scenario. Also, compared to most of the MoMs, Apache Kafka allows delivery of single message to multiple consumers, which significantly simplifies implementation of switch points in the topology. What's also important for the MInTMS is that applications based on Kafka Streams and Connect concepts are scalable by design: if the stream of data can be divided into a number of smaller streams and user runs multiple instances of the same application, these streams will automatically be assigned to different instances – and rebalanced when one of the instances is stopped or a new one added. However, the exact manner in which the stream of data from a main source should be split into smaller streams, as well as the number of these streams, cannot be defined automatically – for it depends on the stream volume and use-case requirements (e.g. limits placed on the execution time for a single regular run of integration process, triggered once in a given period). In other words, it is one of the parameters that can be adjusted by the user defining the integration scenario and should be provided along with the topology description.

Figure 4 provides an overview of Kafka-based MInTMS; components of the scheme are discussed below.

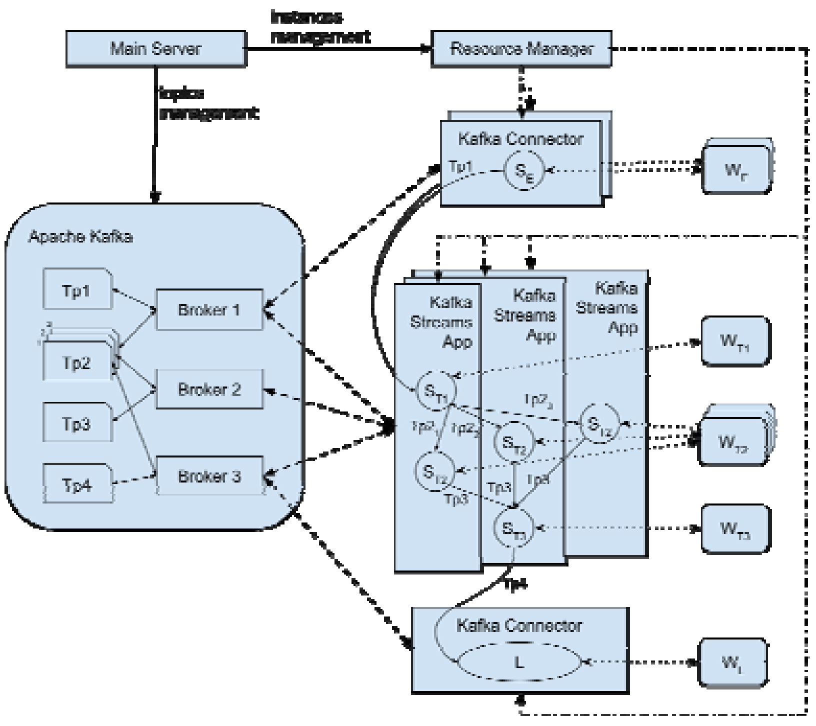

Figure 4. Apache Kafka based architecture of MInTMS.

3.1. Topology Builders

Topology Builder is a component of MInTMS that instantiates a topology: creates nodes and links between them according to the Main Server instructions and scenario description provided by the user. The simplest way to organize the whole integration process topology management, based on the Kafka internals, is to divide it into independent parts, each containing nodes of specific type (E-, T- or L-).

For transformation nodes management can be used Topology class of the Kafka Streams library with a custom wrapper that would configure the Topology instance according to the user's description: create nodes and links between them, initializes State Stores for stateful processing, etc. For extraction and load nodes, Kafka Connector may be used as a Builder for the single-node (sub-)topologies.

Builders implemented this way are capable to operate in distributed mode by design: each instance of a single application is registered with Kafka cluster under the same ID, and Kafka takes care of balancing data flow through the applications so that each instance gets its own part of the whole flow. These sub-flows are mutually exclusive: every message is delivered to exactly one instance.

For Sink Connector (L-Builder) application, which operates with data delivered by Kafka, maximum number of sub-flows and their content are determined by Kafka topic partitions and is based on the *consumer group* concept. When a topic with application input data is divided into a number of partitions and multiple application instances are running, all instances make a consumer group in which each member reads messages from its own "fair share" of partitions. If an instance dies, its partitions are re-assigned to the rest of the group; and if a new instance is registered to the group, partitions distribution is rebalanced. If the number of running application instances is greater than that of the input partitions, excessive applications are starting, but stay idle until one of the operating ones dies – in which case one of the idle ones starts to operate with the partition that previously was assigned to the crushed instance.

Streams application (T-Builder) is much the same: partitions of the topics with input messages determine the maximum level of parallelization of the transformation topology as a whole. In addition, the topology can be divided into sub-topologies by introducing intermediate topics between T-nodes, which may have different number of partitions. In this case each sub-topology instantiated within the Streams application becomes a unit of parallelization with its own degree of parallelism. Due to this, "slow" T-nodes (that require more time for a single record processing) can be grouped into a separate sub-topology with a higher degree of parallelization than that of "fast" ones. In this situation, maximum number of operating (non-idle) Streams application instances will be defined by the number of sub-topologies and their input partitions (see Figure 5). These intermediate topics, if required, are also created by the Main Server when the integration scenario is enabled.

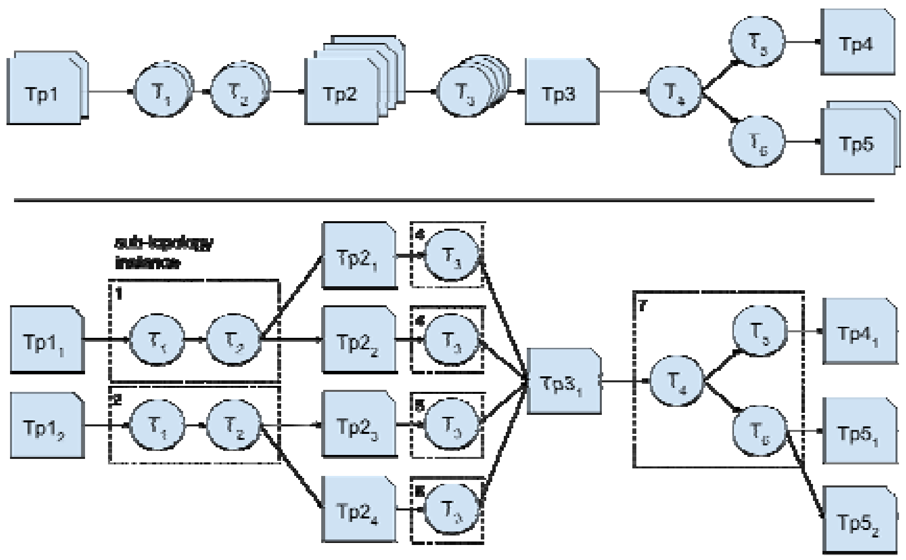

Figure 5. Transformation topology and its division into sub-topologies according to the input and intermediate topics partitioning.

Source Connector (E-Builder), however, is much different to both Sink Connector and Streams Application for it does not have a partitioned input from Kafka topics and has to take care of the partitioning itself. Within the MInTMS context, E-node extracts information from a single main source and represents it as a series of uniform independent messages. The way this process can be parallelized is heavily dependent on the character of the information and its source. It means that Builder, which knows nothing about this yet is responsible for the load balancing, can only provide nodes with information about the number of instances that will be launched and index number (ID) of the given instance. The rest – to understand how to generate its part of messages, mutually exclusive with sets of messages produced by other instances of the same E-node – is up to the worker's implementation, to which mentioned parallelization parameters are passed by the supervisor at the start time.

When the parallelization of the data extraction operations is not required, E-builder can be used in a "singleton" mode in order to simplify the developer's task. In this mode the builder will initialize only one instance of the node, no matter how many instances of the builder are launched. In this case the worker is expected to extract the full set of the required information without any partitioning, while further data flow distribution will be defined by the number of partitions of the Kafka topic, into which messages generated by the E-node are written.

Instantiated by Builders topologies of nodes are linked to each other via Kafka topics. Single-node E-topologies pull data from associated main sources and write them to Kafka topics; these topics are used as input by transformation topologies. Intermediate topics link sub-topologies within the T-topologies, and output topics of T-topologies are used as input for single-node L-topologies. This way the separate topology instances created by Builders are combined into a seamless ETL process that scales in a very flexible manner.

All the required topics are to be created and configured by the Main Server at the moment when the scenario defined by the user is enabled for execution.

3.2. Topology nodes

Nodes supervisors, managed by the Builders, are based on Kafka library classes: SourceTask, Processor and SinkTask. Underlying standard classes provide supervisors with functionality required to communicate with each other via Kafka topics (message queues) and – for SourceTask – manage the main source offset, meaning that Kafka can provide E-nodes with Integration State Storages as well. Supervisors, in turn, extend this out-of-the-box functionality to operate with workers (external processes with user's implementation of the ETL-operation):

- initialize worker according to the starting instructions provided for given topology node;
- monitor the worker's state and keep it alive;
- forward log messages;
- communicate with the worker: trigger worker's operations, send messages for processing and/or receive ones with the processing results;
- handle processing status: in case of a failure initiate reprocessing according to the reprocessing policy and, if message cannot be processed – write it to a dedicated Kafka topic for further investigation by the system administrator and/or to initiate the reprocessing later.

Requirements to the communication protocol between supervisor and worker are much softer than to that between supervisors. It must provide acknowledged point-to-point data transfer – and be as simple as possible. First requirement is essential to guarantee that messages pulled by supervisor from Kafka topic are properly processed and the results are received by the supervisor, while the second is to minimize effort required for workers implementation – it must be possible to easily implement the worker's part in any programming language even from scratch; also, the simpler the protocol – the faster messages can be processed.

However, in some use-cases it may be useful to extend the protocol functionality, e.g. to allow batch processing (in order to avoid querying additional sources for every message or writing integrated information to final storages record by record). To make it possible the protocol should be extendable – which, in turn, makes it more complicated than it could be.

Due to this authors suggest to consider two possible approaches: to develop a custom protocol with minimal service information accompanying useful data (such as that required to distinguish one message from another) or to pick one of standard, widely used protocols like HTTP or a more light-weight STOMP (as long as no message routing is required), extending them with custom headers when needed. The basic version of the custom protocol may be almost primitive, and it will allow fast development of all required ETL modules; however, extending it to the more elaborate version for better flow control would require additional efforts. HTTP- or STOMP-based protocols, on the contrary, can be more difficult for implementation (which can be eliminated by usage of standard or third-party libraries available for many languages) yet more efficient in terms of functionality extension.

3.3. Resource Management

Topology Builders are designed to automatically balance the load between multiple simultaneously running instances. It means that for deployment of multiple instances on distributed resources they can be wrapped in containers, and the Resource Manager functionality – to run these containers on available resources – may be delegated to a special container-orchestration system, such as Kubernetes [4]. Instead of containers, the system may also utilise virtual machines approach and an appropriate VM manager; none of these possibilities, however, were carefully considered at this point of the development and this particular question is yet to be investigated.

4. Real-life experience

MInTMS as an idea originally appeared in the DKB project – Data Knowledge Base for the ATLAS experiment at the LHC [5]. The goal of this project was to provide the ATLAS community with coherent representation of the information about scientific data, used by scientists in their research. The information of interest – in which research the same data were used, data provenance and value in terms of the resources used for their production and storage, etc. – is available from multiple ATLAS information systems, but is highly separated (since each system was developed to address specific tasks and stores only the information required for these tasks). Although the integration of this information into a coherent view is a primary goal of the DKB, due to the uncertainty in the list of information sources involved and what information should be extracted from each source, as an additional requirement to the DKB was stated that it should allow changes in the integration process and provide means to make these changes conveniently. Also, the integration processes within the DKB had to be scalable – since potentially there may be a need to bring together all the metadata of the ATLAS experiment, some of which are generated as fast as half a million of new/updated records per hour (e.g. PanDA jobs [6]).

As a part of the project a prototype of the Kafka-based integration process was developed, utilising the ideas described in the previous sections of this paper [7]. However, the developed prototype was not yet a ready-to-use and fully functional system, requiring further development apart from the main objective of the DKB (ATLAS metadata integration). Due to this – and for the sake of faster result – for the next task within the DKB the integration process was implemented differently: the prototype followed the same concept of logically independent worker nodes (and re-using the case-independent parts of the first prototype), but in the quality of a simplified Builder a set of management tools dedicated to that specific process was created. Although these tools can not be used as-is for any other integration process and do not support distributed execution of the process, they are much less complicated than the corresponding component of the MInTMS. It provided enough functionality to start operating with the integration process and organize regular metadata updates in the final storage.

Currently the integration process operating within the DKB is considered to be fully established and stable enough for the ATLAS community to make use of the metadata provided by the DKB in a trial mode. And at this point for the further improvement of the DKB as a whole (in terms of the automation of administration tasks, health monitoring and fault-tolerance), as well as to introduce the possibility of distributed execution of the integration scenario, it is considered to be sensible to revise the Kafka-based prototype and proceed with its development toward the case-independent MInTMS, described in this paper. This will provide a solid grounding for further development and also reduce time required to address new use-cases in similar information integration tasks.

5. Conclusion

MInTMS is a system designed to assist in the orchestration of metadata integration in different use-cases, not limiting the developers with a predefined set of supported data sources (or source types), allowed operations on data and/or specific programming language for custom operations. Its aim is to simplify common questions, such as process scaling or data delivery control from the original information sources to the final storages with integrated data by encapsulating these tasks into a topology construction framework and leaving to the user only the implementation of the case-specific data operations and the whole process tuning.

The conceptual design of the system core, integration scenario construction framework, can be embodied with the distributed streaming platform Apache Kafka. First steps toward its implementation were made as a part of the DKB project (Data Knowledge Base for the ATLAS experiment at the LHC), both in the Kafka-based version and with simplified case-specific set of tools. Developed prototypes demonstrated the validity and effectiveness of the suggested approach, and the work in this direction will be continued as a part of the efforts to improve the reliability of already operating DKB services and to be applied as a grounding for multiple different analytical tasks in the future – such as aggregation and analysis of accounting information in a supercomputing centre, network usage analytics, etc.

Acknowledgements

This work is supported by Russian Science Foundation under contract №18-37-20003 and by the NRC "Kurchatov Institute" (order No. 1571).

References

[1] Golosova M., Grigorieva M., Aulov V., Kaida A. on behalf of the ATLAS Collaboration. Data Knowledge Base for the ATLAS community. Selected Papers of the 8th International Conference "Distributed Computing and Grid-technologies in Science and Education", pp. 486-492, CEUR Workshop Proceedings, Aachen, Germany (2018).

[2] Apache Kafka homepage, https://kafka.apache.org/, last accessed 2020/02/07.

[3] Fehling C., Leymann F., Retter R., Schupeck W., Arbitter P. Cloud Computing Patterns. Springer, Vienna, Austria (2014). DOI: 10.1007/978-3-7091-1568-8

[4] Kubernetes homepage, https://kubernetes.io/, last accessed 2020/02/07.

[5] ATLAS Collaboration: The ATLAS Experiment at the CERN Large Hadron Collider. Journal of Instrumentation, vol.3(2008), S08003, IOP Publishing, Bristol, England (2008).

[6] Elmsheuser J., Di Girolamo A. Overview of the ATLAS distributed computing system. EPJ Web of Conferences vol.214(2019), 03010, EDP Sciences, France (2019).

[7] Golosova M., Ryabinkin E. Data Management in Heterogeneous Metadata Storage and Access Infrastructures. Selected Papers of the 26th International Symposium on Nuclear Electronics and Computing (NEC2017), CEUR Workshop Proceedings, Aachen, Germany (2017).

Received November 2, 2020.

М.В. Голосова, В.А. Аулов, В.В. Котляр

Разработка архитектуры системы управления топологиями процессов интеграции метаданных.

Препринт отпечатан с оригинала-макета, подготовленного авторами.

Подписано к печати   10.11.2020.   Формат 60 × 84/16.   Цифровая печать.
Печ.л. 1,25.   Уч.– изд.л. 1,8.   Тираж 80.   Заказ   10.   Индекс 3649.

НИЦ «Курчатовский институт» – ИФВЭ
142281, Московская область, г. Протвино, пл. Науки, 1

www.ihep.ru; библиотека http://web.ihep.su/library/pubs/all-w.htm

Индекс 3649

П Р Е П Р И Н Т 2020-5,
НИЦ «Курчатовский институт – ИФВЭ», 2020