Electronic Preprint for Journal of Information Processing Vol.26

Regular Paper

Query Conversion for Aggregate Operations in Distributed SPARQL Query Processing Using Summary Information

Shu Kaneko[1,a) Eiichiro Chishiro[1,b)

Received: January 31, 2018, Accepted: May 12, 2018

Abstract: Resource Description Framework (RDF) is widely used as a framework for uniformly describing resources on the web. RDF expresses various data with triples that consist of subject, predicate and object. Query information can be inquired using the standard query language called SPARQL. Since the amount of retrieval processing also increases for large-scale data, many processing performance improvement methods by distributed processing using multiple servers have been proposed. As for processing of queries that perform aggregations such as SPARQL duplication elimination and average value function, it is impossible to avoid an increase in throughput due to an increase in the amount of data. In this paper, we propose a query conversion processing method so that distributed processing method can be applied to queries including aggregation. In addition, based on a method using summary information of data for distributed processing, the results of the retrieval using the proposed method were evaluated using real data. As a result, we confirmed the speedup of the search process by distributed processing.

Keywords: RDF, SPARQL, query distribution, aggregate queries

1. Introduction

Resource Description Framework (RDF) is standardized by W3C as a unified framework to describe information on the Web [7]. RDF represents each information in a triple, i.e., subject, predicate and object. The whole data are represented by a graph structure (RDF graph) in which the subject is represented by a starting point node, the predicate, by a label to the side edge, and the object, by an end point node. Each information is represented by a Uniform Resource Identifier (URI), or a literal consisting of numerals and character strings. RDF data are stored in a database called an RDF store, and information that matches a query can be retrieved using a standard query language called SPARQL [8].

One of the characteristics of RDF is that it is "schema-less". Since it is not necessary to define the structure of data as a schema beforehand, it is not only easy to change or add data attributes (fields), but also possible to make cross-sectional searches by linking different databases.

Due to these advantages, various data is now represented by RDF, and RDF graphs are becoming increasingly large-scale. Typical examples include DBpedia (about 8.8 billion triples in the English version), which is the RDF version of the Internet encyclopedia Wikipedia, and the knowledge database Uniprot for amino acid sequences in proteins (about 30 billion triples). Since it is necessary to efficiently handle and search such large-scale data, much research has been carried out in this area [3], [5].

Under such circumstances, a distributed processing method for queries using summary information on a RDF data structure has been proposed [1]. A feature of this method is to beforehand generate RDF data summary information called a range graph [*1]. By using a range graph, it is possible to make a distributed processing of queries considering narrowing-down of the scope of solutions and load balancing, resulting in improvement of search efficiency. In fact, it has also been confirmed by a standard benchmark, LUBM [4], that a nearly linear speedup can be obtained. This method is advantageous in that it is easy to introduce since it does not require splitting of RDF data, and that the overhead associated with distributed processing is small since there is no need to repeat fine-grained queries.

On the other hand, in some queries including aggregation, such as aggregation after deduplication or average, there is a problem that it is not always possible to integrate individual results from distributed processing to find a suitable solution.

We are, therefore, investigating ways to solve this problem by generating summary information in such a way that aggregation with deduplication can be performed by efficiently making distributed processing. Results of the proposed method based on a real dataset are promising, and it has been confirmed that processing speed can be increased by a maximum of about 3.41 times in the processing of a query including aggregation.

2. Distributed Processing of Queries Using Summary Information

In this section, distributed query processing of the previous

1 Graduate School of Science and Technology, Seikei University, Musashino, Tokyo 180–8633, Japan
a) dm166202@cc.seikei.ac.jp
b) chishiro@st.seikei.ac.jp

*1 In this paper, the term "RDF data summary information" is used to denote that the information required for the proposed method consists of the triples representing the range graph in paper [1] as well as the correspondence between RDF data and the range graph. However, the core of the summary information is the range graph, and a strict distinction is not required for an understanding of the proposed method.

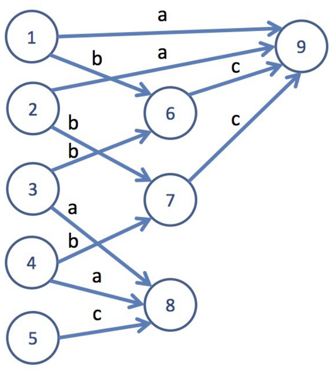

Fig. 1 Target RDF graph.

study [1] based on the master/worker model will be described. In this method, summary information of the RDF graph called a range graph is generated as a pre-process of the search. At the time of search, a first search is performed on the range graph on a master server. Next, distributed processing of queries is performed based on the result, and a partial search result is obtained for each worker server. The search results obtained are then integrated by the master server, and a final search result is generated.

Taking the simple RDF graph shown in **Fig. 1** as an example, the procedure to generate the range graph, and an outline of the distributed search process based thereon, will be described. The RDF graph in Fig. 1 consists of 9 nodes from 1–9, and edges of three predicates, a, b and c. In the standard form, nodes and edges must be URI (IRI), but in this case short names are used to simplify the explanation. Moreover, terms in RDF such as triples, resources, literals, subjects, predicates and objects, and graph terminology such as edges, nodes and edge labels, are used selectively as appropriate depending on the context.

2.1 Generation of Range Graph

The range graph G_D for the RDF graph G is, intuitively, a contracted form obtained by grouping nodes in G based on the identity of the label set of edges connected to that node. The nodes of G_D are configured as equivalence classes of equivalence relationships $\sim$ on G as defined below.

(1) All resources that appear as an edge label (predicate) in the original RDF graph are assumed to be equivalent only to themselves.

$$(\_, n, \_) \in G \Rightarrow n \sim n \wedge (\forall n'. n' \neq n \Rightarrow n' \nsim n).$$

(2) The other nodes n are assumed to be equivalent only when both the input edge label set $I(n)$ and the output edge label set $O(n)$ are the same respectively.

$$(\_, n_1, \_) \notin G \wedge (\_, n_2, \_) \notin G \Rightarrow$$
$$(n_1 \sim n_2 \Leftrightarrow I(n_1) = I(n_2) \wedge O(n_1) = O(n_2)).$$

In the RDF graph, resources used as edge labels (predicates) are sometimes used as nodes (subjects or objects), but in the above definition, all such resources are used as predicates, and are equivalent only to themselves.

In order to save the connection relationship between the nodes in G, the edges of the range graph G_D are configured to meet the following expanded approximation condition.

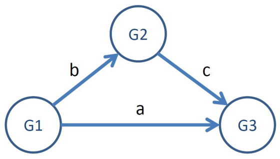

Fig. 2 Range graph.

$$\forall (n_1, p, n_2) \in G.\ \exists t \in G_D.\ t = (\alpha(n_1), \alpha(p), \alpha(n_2)).$$

Here, α is a mapping which maps resources and literals on G, to corresponding resources and literals on G_D, and meets the following conditions.

(1) for a resource p used as an edge label (predicate), $\alpha(p) = p$.

(2) for other resources and literals n_1, n_2, $n_1 \sim n_2 \Rightarrow \alpha(n_1) = \alpha(n_2)$.

The expanded approximation condition ensures that for all edges (triples) (n_1, p, n_2) of G, edges having the same label $(\alpha(n_1), p, \alpha(n_2))$ exist in G_D, and that $\alpha(n_1)$ and $\alpha(n_2)$ correspond respectively to the equivalence classes of n_1 and n_2 [*2].

G_D which satisfies the expanded approximation condition can easily be configured by converting each edge of G (triple) using α. In other words, we may set: $G_D = \{(\alpha(s), \alpha(p), \alpha(o)) | (s, p, o) \in G\}$.

Since the range graph G_D obtained in this way is a homomorphic graph of G, it is possible to obtain an expanded approximation of the search result on the original RDF graph from the search result of the graph pattern on the range graph. For example, when the search result in G of the triple pattern $?x\ p\ ?y$ satisfies $?x = n_x$, and $?y = n_y$, $?x = \alpha(n_x)$, and $?y = \alpha(n_y)$ will always be included in the search results in G_D. In the case where the subject and/or object of the triple pattern are not variables but are resources or literals, queries may be searched after conversion to the corresponding resources or literals on G_D using α.

In addition to the equivalence relation $\sim$ defined above, more equivalence relations that are useful for predicting the range of possible search results are found. In the following, we will give an explanation using the definition that is employed in the method proposed in this paper, and that excludes the condition $I(n_1) = I(n_2)$ related to the input edge label set from condition (2) of the equivalence relation $\sim$.

Figure 2 shows the range graph generated based on the equivalence relationship $\sim$ from the RDF graph in Fig. 1. The correspondence between each node in Fig. 2 and each node in Fig. 1 is as follows [*3]:

G1 = {1, 2, 3, 4}
G2 = {5, 6, 7}
G3 = {8, 9}

The node G1 corresponds to the node set when the label set of the output edges is {a, b}, the node G2 to the node set when the label set of the output edges is {c}, and the node G3 to the node set when the label set of the output edges is an empty set (i.e., output

[*2] Note that $\alpha(\mathtt{p}) = \mathtt{p}$.
[*3] If α is used, it may be expressed for example by $\alpha(1) = \mathtt{G1}$.

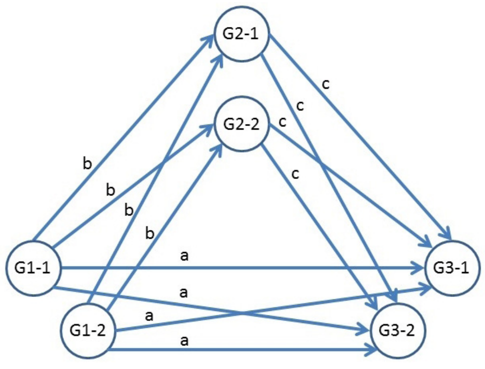

Fig. 3 Summary graph obtained by dividing nodes.

edges do not exist), respectively.

In a previous study [1], each node of the range graph was divided by a specified number of divisions in order to perform distributed processing based on the search results for the range graph. **Figure 3** is a graph obtained by dividing each node of the range graph shown in Fig. 2 by the number 2. Note that in the same way as before, edges are generated so as to satisfy the expanded approximation condition in this division. The correspondence between each node in Fig. 3 and each node in Fig. 1 is as follows:

$\mathtt{G1\text{-}1} = \{1, 2\}$

$\mathtt{G1\text{-}2} = \{3, 4\}$

$\mathtt{G2\text{-}1} = \{5, 6\}$

$\mathtt{G2\text{-}2} = \{7\}$

$\mathtt{G3\text{-}1} = \{8\}$

$\mathtt{G3\text{-}2} = \{9\}$

After generating the range graph, new triples are added representing the correspondence between the nodes of the range graph determined by α, and the nodes of the original RDF graph. For example, the correspondence between the node `G1-1` and the node 1 is represented by a triple (`G1-1`, `abs`, 1) using a new predicate `abs` [*4] representing the correspondence relationship.

2.2 Distributed Search Processing Using Range Graph

In the previous study [1], distributed search processing was performed by the following steps using the range graph obtained by the method described in Section 2.1.

(1) In the master server, a solution of a given query is searched on the range graph, and the node used as an axis of distributed search is determined.

(2) A restriction clause is added to the query, and searched on each worker server.

(3) The results obtained by each worker server are integrated on the master server.

As an example, consider the following query for the RDF graph in Fig. 1.

```
select ?y
where {
  ?x b ?y.
  ?y c ?z.
}
```

This query is to determine the node which is not only the end point of the edge with the label of the predicate `b`, but the start point of the edge with the label of the predicate `c`.

In step (1), a query obtained by converting the constants in the above query by the mapping α, is searched with respect to the range graph of Fig. 2 [*5]. As a result of the search, the following solution is obtained:

$?x = \mathtt{G1}, ?y = \mathtt{G2}, ?z = \mathtt{G3}$

Based on the search results, one variable is selected to be used as a criterion for search range division. In the previous study [1], the variable which, among the nodes of the range graph obtained as a search result, gives the minimum number of nodes of the corresponding original RDF graph as a solution, is selected, on condition that the number of nodes of the RDF graph is equal to or greater than the number of worker servers. The numbers of nodes on the original RDF data associated to each node in Fig. 2 are $|G1| = 4$, $|G2| = 3$, $|G3| = 2$. If the number of worker servers is 2, the variable $?z$ with the solution `G3` is selected, which gives the minimum number of nodes while the number is equal to or greater than the number of worker servers.

In step (2), the following query is generated based on this result. Hereafter, a query generated based on the search result of the range graph will be referred to as a partial query.

```
select ?y
where {
  ?x b ?y.
  ?y c ?z.
  G3 abs ?z.
}
```

Since the partial query includes a triple pattern with predicate `abs` (hereinafter, referred to as a restriction clause), which is added when generating the range graph, the value of $?z$ is restricted to nodes `8` and `9` corresponding to node `G3`. By restricting a different range for each server, it is possible to distribute the search process. For example, when there are 2 worker servers, partial queries where the subject `G3` of the restriction clause of the above query is replaced by `G3-1` and `G3-2`, respectively, are searched on each worker server.

In step (3), by integrating the results of each worker server on the master server, the final result is obtained. In the above example, the search result of a partial query restricted to `G3-1` does not exist. On the other hand, the search result of the partial query restricted to the node `G3-2` is $?y = \{6, 7\}$. By integrating these, the final search result $?y = \{6, 7\}$ is obtained. This coincides with the search result for the original RDF graph.

[*4] For a predicate expressing correspondence, a name must be chosen so as not to conflict with existing predicates.

[*5] If the constant is only a predicate as in this example, the predicate is mapped to itself by α, so the query does not change.

In general, in the case of a query consisting of a triple pattern not including an aggregation, the union of results of distributed searches using restriction clauses always coincides with the search results obtained by the original query. As an example illustrating this, consider the following query.

```
select ?x
where {
 ?x p ?y.
}
```

Now, assume there are triples $(s_1, \mathtt{p}, o_1)$ and $(s_2, \mathtt{p}, o_2)$ which match t. Since the triples,$(\alpha(s_1), \mathtt{p}, \alpha(o_1))$ and $(\alpha(s_2), \mathtt{p}, \alpha(o_2))$ corresponding to these are included in the range graph from the expanded approximation condition, the solution $?x = \{\alpha(s_1), \alpha(s_2)\}$ is obtained by searching on the range graph. Since when generating the range graph, the triples $(\alpha(s_1), \mathtt{abs}, s_1)$ and $(\alpha(s_2), \mathtt{abs}, s_2)$ representing the correspondence between the original RDF graph and the range graph are added, it is clear that the search results of the partial queries q_1 and q_2, which are generated by adding the restriction clauses for $?x$, $\alpha(s_1)$ abs $?x$ and $\alpha(s_2)$ abs $?x$, are identical to the results obtained by searching with the original queries restricted to $?x = s_1$ and $?x = s_2$. Since the same nodes cannot be associated with multiple nodes in the range graph by α, solutions obtained in searches by partial queries are always mutually exclusive. Moreover, due to the expanded approximation condition, the solution to the original query is associated with one of the solutions obtained by searching the range graph. Therefore, the union of those solutions is guaranteed to coincide with the solution of the original query. Without being limited to this example, it is possible to show that this argument is correct for any triple pattern (including multiple patterns).

2.3 Features of Distributed Search Process in Previous Studies

As shown in Section 2.1 and Section 2.2, distributed search is implemented by summary information of the RDF graph called a range graph, triples that represent the correspondence between the nodes of the range graph and the original RDF, and the restriction clause used in searching. These can all be used without changing the existing RDF store implementations, and are easy to use in practice. Also, for some queries, the final result is obtained by only a preliminary search for the range graph, and it is particularly useful for faster search with a query for which there is no solution.

On the other hand, there are also cases where correct results for the original query cannot be obtained by simply integrating the results of the partial queries. For example, consider the case where the following query is given to a graph about videos and tags in **Fig. 4**:

```
select (count(distinct ?t) as ?count)
where {
 ?v tag ?t.
}
```

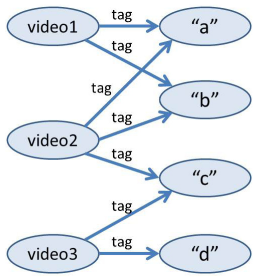

Fig. 4 Graph relating to videos and tags.

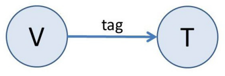

Fig. 5 Summary information for video and tag graph.

This query calculates the number of nodes that are objects of the predicate tag without duplication.

Figure 5 shows the range graph of the RDF graph in Fig. 4. The correspondence between each node in Fig. 5 and each node in Fig. 4 is as follows:

V = {video1, video2, video3}

T = {"a", "b", "c", "d"}

Since the numbers of nodes in the RDF graph in Fig. 4 corresponding to each node of the range graph are $|\mathtt{V}| = 3$, $|\mathtt{T}| = 4$, $?v$ corresponding to V is selected as a variable of the distribution criteria when the number of worker servers is 2. If the nodes corresponding to V are divided by the number 2, and labeled V1 and V2, respectively, they can be assigned as follows:

V1 = {video1, video2}

V2 = {video3}

When distributed search is performed on 2 worker servers for the query with restriction clauses, the worker server for a query with a restriction clause including V1 returns an aggregate value 3 for the value {"a","b","c"} of $?t$, and the worker server for a query with a restriction clause including V2 returns 2 for {"c","d"} of $?t$. However, by adding these results, an aggregate value of 5 is obtained, which is different from the correct number of unique tags, 4.

Similar problems arise in other aggregations such as the average (avg). Thus, in the method of the previous study, for a query that performs aggregations with deduplication, there is the limitation that an efficient distributed search process using summary information cannot be performed correctly.

3. Proposed Method

This section describes a method for solving the problem of aggregation with deduplication in Section 2.3.

In the proposed method, we introduce a new concept called distribution criteria. Distribution criteria are predicates that are specified when summary information is generated, and it is usually assumed that the predicates are frequently used in aggregation.

The aim of the proposed method is to evenly distribute the node set which includes the objects of the specified predicate (the end points of the edges whose label is the specified predicate) such as not to cause duplication when the summary information is generated. Concerning aggregation on the objects with the proposed method, it is unnecessary to perform a process with high overhead such as transferring results before aggregation from the worker server and removing duplication on the master server. The range of queries to be applied can be expanded as necessary by selecting multiple predicates as distribution criteria.

In the following, a description will be given in the order of generation of summary information based on the distribution criteria, judgement of applicability of the proposed method to the query, addition of the restriction clause, and query conversion.

3.1 Generation of Summary Information Based on Distribution Criteria

As described above, in the proposed method, summary information is generated in such a way that aggregation with deduplication for an object of a predicate specified as the distribution criteria, can be computed efficiently by distributed processing.

The flow of generating summary information is the same as the procedure described in Section 2.1 up to node division. At node division, in the proposed method, the nodes corresponding to the object of the predicate specified as the distribution criteria are preferentially divided. It is desirable that the number of divisions, although it is arbitrary, is equal to or more than the number of worker servers as long as possible.

Next, triples that represent the correspondence between the nodes of the range graph and the nodes of the original RDF graph are added. The nodes are associated avoiding duplication while balancing the sizes of the nodes sets of the original RDF graph corresponding to each node of the range graph after the previous division.

Consider Fig. 4 used in Section 2.3, and a query which is used to find the number of unique objects of a predicate `tag`, as an example. It is assumed that the number of worker servers is 2, and the predicate `tag` is specified as the distribution criterion.

By the same procedure as in Section 2.1, the range graph of Fig. 5 is obtained. Here, `T`, which is the object of the predicate `tag` taken as the distribution criterion, is divided into 2 parts since the number of worker servers is 2. The result of the division is shown in **Fig. 6**.

Next, triples representing the correspondence between the range graph and the original RDF graph are added. As described above, while ensuring no duplication, they are associated so that the sizes of the node sets of the original RDF graph corresponding to the nodes `T1` and `T2` of the range graph after division are as even as possible. **Figure 7** shows the result of adding the triples representing the correspondence relationship.

For an RDF graph to which such triples are added, there is no duplication in the search results for queries to which restriction clauses related to `T1` and `T2` are added. Thus, correct results can be obtained without performing deduplication.

For the handling of aggregation queries with deduplication, it is conceivable that each worker server forwards the result set to the

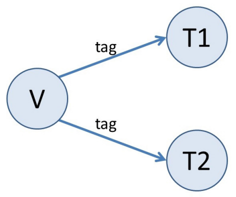

Fig. 6 Summary information obtained by dividing nodes.

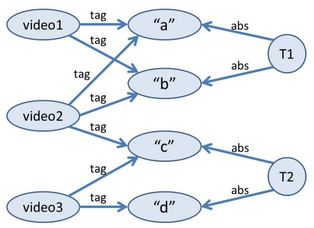

Fig. 7 Graph with addition of summary information and correspondence relation.

master server before aggregation, and the master server performs aggregation after deduplication. But if the result set before aggregation is large, the transfer takes time. In the proposed method, it is unnecessary to transfer the result set before aggregation, so it can be expected to greatly shorten the processing time for such a query.

3.2 Query Judgement and Adding Restriction Clause

First, a method for determining whether or not a given aggregation query with a `distinct` qualifier is a target of the proposed method, will be described. As the range graphs generated based on the method described in Section 3.1 satisfy the expanded approximation condition described in Section 2.1, queries that can be handled in the previous study [1] can be handled similarly.

Briefly, a target query has an aggregation in the `select` clause and the target variable of the aggregation is specified in the `where` clause as the object of the predicate of the distribution criterion.

For example, in a query relating to the number of unique tags shown in Section 2.3, the variable $?t$ is specified in the `select` clause. Focusing on this variable, and looking at the triple pattern $?v$ `tag` $?t$ in the `where` clause, $?t$ is actually used as the object of the predicate `tag`. Therefore, if the predicate `tag` is specified as the distribution criterion, the proposed method can be applied to this query.

The proposed method is applicable only to the queries in which one variable is specified in the `select` clause. For this reason, queries that include two or more `distinct` qualifier variables as shown below are excluded from this method.

```
select
 (count (distinct ?v as ?count_video)
 (count (distinct ?t as ?count_tag)
where {
 ?v tag ?t.
}
```

Next, a method of adding a restriction clause to a query judged to be within the scope of this method, will be described. The difference from the procedure shown in Section 2.2 is the method

Table 1 Combinations of aggregation and solution qualifiers.

aggregation	No qualifiers	distinct	reduce
count	○	*	-
sum	○	*	-
avg	*	*	-
min	○	○	-
max	○	○	-
groupconcat	-	-	-
sample	-	-	-

of selecting the variable taken as a criterion for the search range division. In order to avoid the problem of duplication, it is necessary to select the target variable of aggregation (?*t* in the above example) irrespective of the number of worker servers and the number of corresponding nodes.

Note that regarding the `distinct` qualifier in the `select` clause, search range division is performed such that there is no duplication. Thus, if the `distinct`-qualified variable appears only as an object of the predicate of the distribution criterion in the `where` clause, it is also possible to remove the `distinct` qualifier. However, if another triple pattern which has this variable as a subject or object is contained in the query, the `distinct` qualifier cannot be removed because there is a possibility that duplication may occur due to them.

3.3 Query Conversion

In this section, based on the discussion heretofore, a method of converting a query which performs aggregation, to a partial query which can be handled by distributed processing, will be described. According to the SPARQL Query Language Specification [8], there are 7 types of aggregations as shown in **Table 1**, and there are 2 types of solution qualifiers that can be combined, i.e., `distinct` and `reduce`. Of these, `groupconcat`, `sample` and `reduce` qualifiers are not much used in practice, and therefore are not discussed in this paper.

Table 1 summarizes the required query conversion for each type of aggregation and qualifier combination. In the table, ○ indicates that the original query can be used as it is as a partial query, and * indicates the case where conversion processing is required for the original query. For example, if `count` is specified as an aggregation without a qualifier, it is only necessary to simply add up the numbers obtained from each worker server on the master server, and this is indicated by ○.

In the distributed search, it is assumed that the target dataset includes summary information such as range graphs in which deduplication etc. have been performed in advance as shown in Section 3.1, and each worker server holds the same dataset. On the other hand, since the addition of the restriction clause necessary for using the summary information is the same as in the previous section, it is omitted in the conversion rule.

In the following, the conversion method is described for each aggregation.

3.3.1 `count` and `sum`

For `count` and `sum` without a `distinct` qualifier, there is no need to perform query conversion since the aggregation results of each worker server can be summed up on the master server.

Even if there is a `distinct` qualifier, it is possible to remove the `distinct` qualifier since deduplication problems can be solved by a restriction clause as described in Section 3.1.

3.3.2 `min` and `max`

For `min` and `max`, the presence or absence of the `distinct` qualifier does not affect the result due to the nature of the operation. Therefore, there is no need to perform a special query conversion, and integration is possible on the master server by taking the minimum value or the maximum value of the results aggregated on each worker server.

3.3.3 `avg`

In the case of `avg`, application of a conversion rule is required regardless of the presence or absence of the `distinct` qualifier.

If there is no `distinct` qualifier, the number of elements and sum may be calculated on each worker server, and the average can be calculated from these results at the time of integration. The conversion rule is shown below.

Before conversion

`select` (`avg` (?x) `as` ?*avg*)
`where` Q

After conversion

`select` (`count` (?x) `as` ?*count*)
  (`sum` (?x) `as` ?*sum*)
`where` $\hat{Q}$

Here, $\hat{Q}$ represents the `where` clause Q added of a restriction clause.

At integration, the values of the results aggregated by each worker server (?*count* and ?*sum*) may be totaled, and the latter may be divided by the former.

If there is a `distinct` qualifier, the problem of deduplication can be resolved by a restriction clause as in Section 3.3.1, so the above process may be performed after removing the `distinct` qualifier.

4. Evaluation Test

In order to evaluate the effectiveness of the proposed method, the time required for aggregation is measured using the NicoNico Dataset [2]. The NicoNico Dataset is a dataset released to researchers by the National Institute of Informatics to which Dwango Co., Ltd. and Mirai Kensaku Brazil provided the dataset. In this experiment, about 14 million videos (posting date, title, tags, etc.), from March 6, 2007 to August 31, 2016 were converted to RDF, and the data corresponding to about 67,000 videos was used. The scale of the dataset is about 500 million triples, and its size on disk is about 77 GB. The test was performed in a master/worker environment with 4 worker servers having CPU: IntelCore i7 930 2.8 GHz, OS: CentOS 7.3.1611, and memory: 10 GB.

The aggregations used for the test were the following two types.

(1) Number of unique tags
(2) Number of unique comments

A summary of the dataset structure is shown in **Fig. 8**. Summary information was generated using the predicate `tag` representing a tag, and the predicate `content` representing a com-

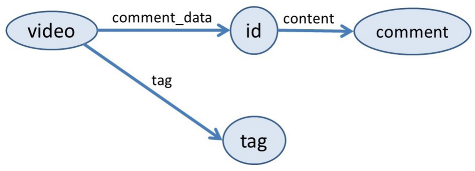

Fig. 8 Data structure of video, tags and comments in video information.

(1) number of unique tag

Original query

```
select (count (distinct ?t) as ?count)
where {
 ?v tag ?t.
}
```

Query after conversion

```
select (count (?t) as ?count)
where {
 ?v tag ?t.
 T1 abs ?t.
}
```

(2)number of unique comment

Original query

```
select (count (distinct ?c) as ?count)
where {
 ?id content ?c.
}
```

Query after conversion

```
select (count (?c) as ?count)
where {
 ?id content ?c.
 C1 abs ?c.
}
```

Fig. 9 Aggregation queries used for evaluation.

ment, as distribution criteria, among the predicates contained in the dataset. On the range graph, the object node of the predicate `tag` is represented by `T`, and the object node of the predicate `content` by `C`. Since the number of worker servers was 4, the division number was is also set to 4. That is, the node `T` was divided into `T1`~`T4`, and the node `C` was divided into `C1`~`C4`.

Figure 9 shows the queries, and queries after conversion, used for the search. In the post-conversion queries, different restriction clauses are added for each worker server, but only the post-conversion queries for one of the worker servers are shown here.

As the result of each query, the number of unique tag is 158,513, and the number of unique comment is 31,379,291. This coincides with the result of removing duplicates from the totals of 440,679 and 89,453,432, respectively.

Table 2 and **Table 3** show the execution times for each query according to whether or not there is summary information, and when the number of worker servers is changed, respectively. The measurement value is the time required for the master to send the query to each worker server until the master finishes integrating the results. Since the execution time could be shorter than the first run when the dataset was on the disk cache, we executed multiple runs until the execution time got stable and then average execution time of 3 runs was measured.

Table 2 Number of unique tag.

	Run time
With summary information (no. of worker servers: 4)	1.75 sec
With summary information (no. of worker servers: 2)	1.86 sec
With summary information (no. of worker servers: 1)	2.34 sec
Without summary information	1.66 sec

Table 3 Number of unique comment.

	Run time
With summary information (no. of worker servers: 4)	195.92 sec
With summary information (no. of worker servers: 2)	285.73 sec
With summary information (no. of worker servers: 1)	585.03 sec
Without summary information	668.22 sec

Table 4 Run time for each worker server.

	Run time
Worker server 1	150.32 sec
Worker server 2	166.10 sec
Worker server 3	153.10 sec
Worker server 4	187.55 sec

In Table 2 for the query that aggregates the number of unique tags, it takes more time for distributed processing. This may be due to communication overheads, etc. since the queries are with a small search range and less results although the dataset itself is relatively large-scale data. For Table 3 in which the number of unique comments is aggregated, the scale is about 200 times the search results related to number of unique tags, and the effect of using summary information and distributed processing is very significant. Regarding the change in the number of worker servers, there is an increase in speed of approximately 2.05 times from 1 server to 2 servers, and approximately 1.46 times from 2 servers to 4 servers. Further, regarding run time with 4 worker servers, there is an increase in speed of approximately 3.41 times using the summary information compared to the speed without using it.

Table 4 shows, in the case of 4 worker servers, the average run time of each worker server when a query which aggregates the number of unique comment is executed.

The difference of run time between worker servers is at most 37.23 seconds, approx. 19.00% compared to the total run time, which shows there is a large scatter. As described in Section 3.1, the search range is allocated to the workers uniformly, but since there are literals containing relatively long character strings, such as video comments which are aggregation targets of this query, there is a possibility that load balancing of the process was insufficient. A possible solution to this problem might be dividing the task more finely than the number of worker servers, and allocating processing on workers dynamically.

5. Related Research

As a distributed processing method for RDF stores, a method of dividing and arranging the data itself in each worker has been proposed [5]. An advantage of data division is that the performance required of the worker in terms of memory and disk size can be relaxed since the data size retained in the worker server can be reduced. However, due to communication and synchronization overheads during the search, it has been pointed out that performance may be degraded for complex queries.

In addition, a method of placing RDF graphs on HBase or MySQLCluster using the MapReduce framework has been proposed [3]. In this method, search is performed by iteratively processing finely decomposed queries, but it has been pointed out that processing time increases when many triple patterns are included.

6. Conclusion

In this research, we focused on the distributed search method for large-scale RDF using summary information, especially on aggregation and deduplication, and made proposals regarding generation of summary information and query conversion. The results of the evaluation test are promising, and the usefulness of the proposed method was confirmed with the Niconico Dataset. The proposed method can be applied to processing where calculation efficiency is a problem in practical applications, such as computation of co-occurrence information of tags and comments on the Niconico Dataset in an existing study [6], and we believe that various applications can be expected in the future.

On the other hand, the proposed method is still in the process of development, and there are still many issues to be considered. For example, in the aggregation involving deduplication, in the proposed method, priority is given to reducing the amount of data transfer from the worker server to the master server, and then variables that become the distribution criteria are selected. However, depending on the query and the dataset, there are cases where it is better to select the variables for distribution criteria by prioritizing load balancing, etc., and to perform duplication processing by the master server as before. If the number of solutions in each worker server can be estimated using the summary information of the proposed method, it may be possible to make the best choice taking both possibilities into account, and we will consider this in the future.

In addition, the Niconico Dataset tested on this research has a simple graph structure, and does not take advantage of the characteristic of RDF that it is possible to express irregular information flexibly. In order to verify the practicality of the proposed method, it is necessary to evaluate a dataset with the characteristics of RDF, and we plan to conduct this in the future. At that time, by taking advantage of the benefits of RDF, we also want to consider methods of using summary information when searching across multiple RDF data (for example, DBpedia and tags in and Niconico Video).

References

[1] Chishiro, E., Miyata, Y., Yokoi, K. and Nishiyama, H.: Distributed SPARQL Query Processing Based on Range Partitioning, *Computer Software*, Vol.30, No.3, pp.180–186 (2013).

[2] Dwango Co., Ltd., Mirai Kensaku Brazil and National Institute of Informatics: NicoNico Dataset, available from ⟨https://www.nii.ac.jp/dsc/idr/nico/nico.html⟩.

[3] Franke, C., Morin, S., Chebotko, A., Abraham, J. and Brazier, P.: Distributed semantic web data management in HBase and MySQL cluster, *2011 IEEE International Conference on Cloud Computing* (*CLOUD*), pp.105–112, IEEE (2011).

[4] Guo, Y., Pan, Z. and Heflin, J.: LUBM: A benchmark for OWL knowledge base systems, *Web Semantics: Science, Services and Agents on the World Wide Web*, Vol.3, No.2, pp.158–182 (2005).

[5] Harris, S., Lamb, N. and Shadbolt, N.: 4store: The design and implementation of a clustered RDF store, *5th International Workshop on Scalable Semantic Web Knowledge Base Systems* (*SSWS2009*), pp.94–109 (2009).

[6] Sakaji, H., Kohana, M., Kobayashi, A. and Sakai, H.: Estimation of Tags via Comments on Nico Nico Douga, *2016 19th International Conference on Network-Based Information Systems* (*NBiS*), pp.550–553, IEEE (2016).

[7] W3C: RDF Primer, available from ⟨http://www.w3.org/TR/rdf-primer⟩.

[8] W3C: SPARQL 1.1 queryLanguage, available from ⟨https://www.w3.org/TR/2013/REC-sparql11-query-20130321/⟩.

Shu Kaneko 2018 Completed Master course at the Department of Information Science, Graduate School of Science and Engineering, Seikei University. While at the University, he conducted research on databases using RDF. He is currently engaged in rail cargo transport at Japan Freight Railway Company.

Eiichiro Chishiro 2009 Graduated from the Doctoral Course at the Graduate School of Information Science and Technology, the University of Tokyo. After working as a researcher at Yokohama Research Laboratory, Hitachi, Ltd., and then assumed the present post. He is a doctor of Information Science and Technology, and is now engaged in research and development of compilers, program analysis/verification, graph databases, and other topics.