

Top-k Pipe Join

Davide Martinenghi, Marco Tagliasacchi

Dipartimento di Elettronica e Informazione – Politecnico di Milano
Piazza Leonardo da Vinci, 32 – 20133 Milano, Italy
{martinen, tagliasa}@elet.polimi.it

Abstract—In the context of service composition and orchestration, service invocation is typically scheduled according to execution plans, whose topology establishes whether different services are to be invoked in parallel or in a sequence. In the latter case, we may have a configuration, called *pipe join*, in which the output of a service is used as input for another service. When the services involved in a pipe join output results sorted by score, the problem arises of efficiently determining the join tuples (aka *combinations*) with the highest combined scores. In this paper we study different execution strategies related to the pipe join configuration. First, we consider a strategy that minimizes the access costs to achieve a target number of combinations. Then, we propose a strategy that explicitly considers the scores of the output tuples in order to provide deterministic guarantees that the top-k combinations have been found. Finally, a hybrid strategy is presented.

I. SCENARIO

We consider a pipe join between two services s_L and s_R , where (part of) the output of s_L (the “left” service) is used as input for s_R (the “right” service). W.l.o.g., we assume both services to have the signature $s_i(A_i, B_i, S_i)$, $i \in \{L, R\}$, where A_i is an attribute specific of service s_i , B_i is an attribute used for the join between s_L and s_R , and S_i is an attribute carrying the score for the tuples output by service s_i ¹.

We assume that s_L enables *sorted access* (i.e., tuples are output one by one in descending order of the score S_L), whereas s_R enables *incremental random access* on the join attribute B_R . This means that, for a given value of the attribute B_R , the corresponding tuples are output one by one in descending order of the score S_R . In addition, for each join attribute value, random access can be paused at a given depth and resumed any time later until the tuples are exhausted. For simplicity, in the following we assume that the score of a combination is given by the sum of S_L and S_R , but any other monotonic function would apply.

These are the natural access modes that arise in the context of search computing, where answering complex queries might require invoking heterogeneous services, each characterized by its own access pattern. Consider, as a motivating example, querying for the best combinations of hotels and restaurants in Paris, located in the same district. Service s_L might return all the restaurants sorted by score, whereas s_R needs to receive as input a given district and outputs restaurants sorted by score.

Note that the pipe join configuration studied in this paper differs from the nested loop rank-join introduced by Ilyas et

¹The framework effortlessly adapts to the case where A_i and B_i are sets of attributes instead of single attributes.

al. in [1]. In fact, in the latter case, for each of the tuples of s_L all the tuples of s_R are read sorted by score, i.e. only sorted access is available for s_R . Conversely, here we assume that s_R enables incremental random access on the join attribute B_R .

Previous works in top-k query processing are surveyed in [2]. The main contributions of the paper are strategies for executing a pipe join with the goal of finding the best combinations while minimizing the incurred access cost. Each strategy decides, at each step, whether to access s_L or s_R , and, in the latter case, what input value should be used. In particular, in Section II we discuss an *any-k* pipe join execution strategy, which merely aims at minimizing the cost of accessing the services to obtain k combinations. In Section III we describe a *top-k* strategy that considers the actual scores of the tuples extracted from s_L and s_R in order to determine the top-k combinations and to decide the next move. In Section IV we describe a hybrid *any-k* and *top-k* strategy that combines both aforementioned strategies. Finally, in Section V we illustrate some preliminary experimental results on synthetic data.

II. ANY-K EXECUTION STRATEGY

In our context, the access to services is the main cost indicator, as dictated by network links and transfer rates. We therefore associate every service and access mode with a cost, so as to take into account the different response times for s_L and s_R as a driving factor for optimization of the overall response time incurred by the pipe join. To this end, we provide a characterization of the two services in terms of the following components:

- c_s : cost of a sorted access to s_L .
- c_r : cost of a random access to s_R .
- N : the cardinality $|s_L|$ of the extension s_L of s_L , i.e. the relational set of all tuples that can be returned by s_L .
- J : number of distinct values for B_L in s_L .
- Q : average number of times the same value for B_L occurs in s_L .

In the following, we are going to describe strategies for accessing the tuples available at s_L and s_R . We therefore also introduce the following notation so as to refer to the “current” state of execution of a pipe join.

- n_L : number of tuples fetched from s_L .
- b^j : the j th value for the join attribute B_L ($j = 1, \dots, J$).
- q^j : number of tuples fetched from s_L with $B_L = b^j$. We have $\sum_{j=1}^J q^j = n_L$.
- n_R^j : number of tuples fetched from s_R with $B_R = b^j$.

The number of combinations (hereafter also denoted *any-combinations*) formed in a given state is given by

$$\mathcal{K}(n_L, n_R^1, n_R^2, \dots, n_R^J) = \sum_{j=1}^J q^j n_R^j \quad (1)$$

and the corresponding incurred cost is

$$\mathcal{C}(n_L, n_R^1, n_R^2, \dots, n_R^J) = c_s \cdot n_L + c_r \sum_{j=1}^J n_R^j \quad (2)$$

Under the simplifying assumption that $E[q^j] = \frac{Q}{N} n_L, \forall j$, the expectation of (1) is given by

$$\bar{\mathcal{K}}(n_L, n_R^1, n_R^2, \dots, n_R^J) = E[\sum_{j=1}^J q^j n_R^j] = \frac{Q}{N} n_L \sum_{j=1}^J n_R^j \quad (3)$$

In this section, we address the following optimization problem,

$$\begin{aligned} & \text{minimize} && \mathcal{C}(n_L, n_R^1, n_R^2, \dots, n_R^J) \\ & \text{subject to} && \bar{\mathcal{K}}(n_L, n_R^1, n_R^2, \dots, n_R^J) \geq k \\ & && n_L, n_R^j \in \mathbb{N} \end{aligned} \quad (4)$$

Relaxing the integer constraint, the problem

$$\begin{aligned} & \text{minimize} && c_s \cdot n_L + c_r \sum_{j=1}^J n_R^j \\ & \text{subject to} && \frac{Q}{N} n_L \sum_{j=1}^J n_R^j \geq k \end{aligned} \quad (5)$$

is easily solved, due to its symmetry. The optimal solution satisfies $\frac{n_L^*}{\sum_{j=1}^J n_R^{j,*}} = \frac{c_r}{c_s}$. This result suggests a straightforward stochastic execution strategy.

Algorithm AkPJ (any-k pipe join)

- 1) Initialization: Perform one sorted access to s_L .
- 2) At each step
 - With probability $p_L = \frac{c_r}{c_s + c_r}$ access s_L .
 - With probability $p_R = 1 - p_L$ access s_R using a value b .

The join attribute value b can be chosen in different ways:

- *Round-robin*: Select b^1, b^2, \dots, b^J , then cycle back.
- *Uniform sampling*: Sample b^j with uniform probability $p^j = 1/J$.
- *Histogram-based sampling*: Sample b^j with non-uniform probability q^j/n_L , i.e. equal to the fraction of tuples fetched from s_L with $B_L = b^j$.

Both round-robin and uniform sampling have the drawback of possibly accessing s_R with a join attribute value that has not been discovered in s_L yet. Thus, no new combinations can be formed after the random access. Conversely, histogram-based sampling is an adaptive strategy that maximizes the expected number of new combinations that can be formed after the random access to s_R .

III. TOP-K EXECUTION STRATEGY

Note that AkPJ is a naive strategy that is unaware of the scores of the tuples output by the two services. As such, AkPJ cannot guarantee that the set of combinations found contains the set of top-k combinations. Addressing the top-k pipe join problem requires guaranteeing that the k combinations with highest score (henceforth *top-combinations*) are

extracted. Similarly to the conventional rank-join problem [3], where both services enable sorted access, we need to devise a *bounding scheme* and a *pulling strategy*. The bounding scheme provides an upper bound on the aggregated score of the unseen combinations. For the top-k pipe join problem, it is determined by a threshold value τ computed as

$$\tau = \max\{\tau_L, \tau_R\}, \text{ where} \quad (6)$$

$$\tau_L = S_L(n_L) + \max_{j=1, \dots, J} \{S_R^j(1)\}, \quad (7)$$

$$\tau_R = \max\{\tau_R^1, \dots, \tau_R^J\}, \quad (8)$$

$$\tau_R^j = S_L(n_L^j) + S_R^j(n_R^j), \text{ for } 1 \leq j \leq J. \quad (9)$$

Here, $S_L(i)$ represents the score of the i -th tuple extracted from s_L , $S_R^j(i)$ the score of the i -th tuple extracted from s_R such that $B_R = b^j$, and n_L^j is defined as the position of the first tuple output by s_L for which $B_L = b^j$ or n_L if there is no such tuple among the first n_L . The underlined version $\underline{S}_R^j(1)$ is defined as “if $n_R^j \geq 1$ then $S_R^j(1)$ else S_R^{\max} ”, where S_R^{\max} is the maximum score possible for tuples in s_R . We also define $S_R^j(0) = S_R^{\max}$. A combination is guaranteed to be in the set of *top-combinations* if its aggregated score is greater than τ . An optimal pulling strategy minimizes the cost of accessing the services while guaranteeing that the top-k combinations have been found. Similarly to conventional rank-join, we propose the following strategy based on the threshold definition above.

Algorithm TkPJ (top-k pipe join)

- 1) At each step
 - If $\tau_L \geq \tau_R$ access s_L
 - If $\tau_L < \tau_R$ access s_R using the join attribute $b^j, j = \arg \max_j \tau_R^j$

It is possible to show that in the absence of ties in the aggregated scores TkPJ defines a strategy that minimizes the access cost. Note that such a strategy does not depend on the, possibly heterogeneous, costs of the services involved (c_s and c_r). In fact, in order to find the top-k combinations the execution of the pipe join needs to achieve some minimum depths $n_L^+, n_R^{j,+}, j = 1, \dots, J$, that are independent from c_s and c_r .

IV. HYBRID EXECUTION STRATEGY

In this section we present a hybrid execution strategy that considers both access costs and score distributions. The proposed greedy strategy tries to minimize the cost of reaching a desired number of combinations expressed as a weighted sum of *any-combinations* and *top-combinations*. At each step, we determine the service to invoke (i.e. s_L or s_R) and, when invoking s_R , the value used for the random access. To this end, for each of the $J + 1$ possible choices, we estimate the expected number of: i) new *any-combinations* $\Delta\mathcal{K}_{any}$; ii) new *top-combinations* combinations $\Delta\mathcal{K}_{top}$.

A. Fetching a new tuple from s_L

Let w^j denote the probability that the next tuple will contain the join attribute b^j , which can be estimated as q^j/n_L . The

expected number of new *any-combinations* formed accessing one tuple from s_L is

$$\Delta\mathcal{K}_{any} = \sum_{j=1}^J w^j n_R^j \quad (10)$$

Fetching a tuple from s_L affects $\Delta\mathcal{K}_{top}$ ($\leq \Delta\mathcal{K}_{any}$) in two ways:

- 1) The threshold τ might decrease. Thus, some number of *any-combinations* formed so far might become *top-combinations*. Let this number be $\Delta\mathcal{K}_{top,a}$.
- 2) A fraction of the new *any-combinations* formed (i.e. a fraction of $\Delta\mathcal{K}_{any}$) might have an aggregated score greater than τ , so that they become *top-combinations*. Let this number be $\Delta\mathcal{K}_{top,b}$.

In order to estimate $\Delta\mathcal{K}_{top,a}$ and $\Delta\mathcal{K}_{top,b}$, we need an estimate $\hat{S}_L(n_L+1)$ of $S_L(n_L+1)$. We adopt a simple linear predictor

$$\hat{S}_L(n_L+1) = 2S_L(n_L) - S_L(n_L-1) \quad (11)$$

that extrapolates the value of the last two scores, with which a new value $\hat{\tau}$ for the threshold can be predicted. Then $\Delta\mathcal{K}_{top,a}$ can be immediately determined. As for $\Delta\mathcal{K}_{top,b}$, for each join attribute b^j , we determine how many new combinations formed with the fetched tuple will be *top-combinations*, and we take the expectations by averaging with weights given by w^j .

B. Fetching a new tuple from s_R

The number of new *any-combinations* that can be formed with a new tuple from s_R fetched via a random access with the join attribute value b^j is

$$\Delta\mathcal{K}_{any}^j = q^j \quad (12)$$

$\Delta\mathcal{K}_{top,a}^j$ and $\Delta\mathcal{K}_{top,b}^j$ are determined in a similar way as before, via a prediction $\hat{S}_R^j(n_R^j+1)$ of the score of the next accessed tuple using b^j .

C. Determining the next step

Let us define:

$$\mathcal{J}_L = \frac{\alpha(\Delta\mathcal{K}_{top,a} + \Delta\mathcal{K}_{top,b}) + (1-\alpha)\Delta\mathcal{K}_{any}}{c_s}, \quad (13)$$

$$\mathcal{J}_R^j = \frac{\alpha(\Delta\mathcal{K}_{top,a}^j + \Delta\mathcal{K}_{top,b}^j) + (1-\alpha)\Delta\mathcal{K}_{any}^j}{c_r}, \quad (14)$$

where $\alpha \in [0,1]$ is a tunable parameter that enables to gracefully switch from a strategy that considers only the number of *any-combinations* ($\alpha = 0$) to one that considers only the number of *top-combinations* ($\alpha = 1$). These ratios suggest another stochastic execution strategy.

Algorithm ATkPJ (any- and top-k pipe join)

- 1) Initialization: Perform one sorted access to s_L .
- 2) At each step
 - With probability

$$p_L = \frac{\mathcal{J}_L}{\mathcal{J}_L + \sum_{j=1}^J (p_R^j / \sum_{i=1}^J p_R^i) \mathcal{J}_R^j}$$

$$= \frac{\mathcal{J}_L}{\mathcal{J}_L + \left(\sum_{j=1}^J (\mathcal{J}_R^j)^2 \right) / \left(\sum_{j=1}^J \mathcal{J}_R^j \right)}$$

access s_L .

- With probability

$$p_R^j = (1-p_L) \frac{\mathcal{J}_R^j}{\sum_{j=1}^J \mathcal{J}_R^j}$$

access s_R using join attribute value b^j .

V. SIMULATED EXECUTION AND EXPERIMENTS

Let us consider the simple example in Table I. At the current state of execution, $n_L = 4$, $n_R = 2$, $n_R^2 = 3$, $n_R^3 = 1$. The total number of *any-combinations* that can be formed is equal to 9, and they are shown in Table I(e). The current value of the threshold is $\tau = \max(\tau_L, \tau_R^1, \tau_R^2, \tau_R^3) = 1.6$, where $\tau_L = 0.5 + \max(0.8, 1.0, 0.5) = 1.5$, $\tau_R^1 = 1.0 + 0.5 = 1.5$, $\tau_R^2 = 0.9 + 0.7 = 1.6$ and $\tau_R^3 = 0.5 + 0.5 = 1.0$. Thus, 6 combinations have an aggregated score that exceeds the threshold and thus can be declared to be top-6 (in blue in Table I(e)).

There are $J+1 = 4$ different options that need to be considered: fetching a tuple from s_L , or fetching a tuple from s_R using, respectively, the join attribute value b^1 , b^2 and b^3 . Let us assume that the access costs are $c_s = 1$ and $c_r = 2$. Given the observed join attribute values in s_L , we set $w_1 = \frac{1}{4}$, $w_2 = \frac{1}{2}$, and $w_3 = \frac{1}{4}$. We note that such values can be modified if a prior distribution of the join attribute values is known.

If the simple *AkPJ* strategy described in Section II is adopted, at the next step we fetch a tuple from s_L (s_R) with probability $p_L = \frac{c_r}{c_s+c_r} = \frac{2}{3}$ ($p_R = \frac{c_s}{c_s+c_r} = \frac{1}{3}$). If s_R is selected, the join attribute value used for the random access can be determined using histogram-based sampling, i.e. $p_R^1 = \frac{1}{3}w_1 = \frac{1}{12}$, $p_R^2 = \frac{1}{3}w_2 = \frac{1}{6}$, $p_R^3 = \frac{1}{3}w_3 = \frac{1}{12}$.

Instead, if the *TkPJ* strategy described in Section III is adopted, at the next step we fetch a tuple from s_R (since $\tau_L = 1.5 < \tau_R = 1.6$) using the join attribute value b^2 (since $\tau_R^2 = \max\{\tau_R^1, \tau_R^2, \tau_R^3\}$).

Finally, in order to determine the next step for the hybrid strategy described in Section IV, we need to compute the various quantities shown in Table I(f). By setting $\alpha = 0.5$, with probability $p_L = 0.5385$ it will access s_L , and with probability $p_R = 0.4615$ it will access s_R ($p_R^1 = 0.0659$, $p_R^2 = 0.3297$, $p_R^3 = 0.0659$).

We now compare AkPJ with the simpler TkPJ by means of simulations on synthetic data, generated for both services with $J_i = 10$ distinct join attribute values (all of them are in common between the services). For each join value, we generate a number of tuples. In order to simulate a realistic scenario,

TABLE I

TUPLES (BLACK: RETRIEVED, GREY: NOT RETRIEVED) IN (A) s_L ; (B) s_R WITH $B_R = b^1$; (C) s_R WITH $B_R = b^2$; (D) s_R WITH $B_R = b^3$. (E) COMBINATIONS THAT CAN BE FORMED WITH THE CURRENT STATE OF EXECUTION. IN BLUE, THE SCORES OF THE TOP-6 COMBINATIONS DETERMINED BASED ON THE CURRENT THRESHOLD $\tau = 1.6$. (F) PARAMETERS COMPUTED BY THE PROPOSED ALGORITHM AT THE CURRENT STATE OF EXECUTION.

(a)			(b)			(c)			(d)			(e)				(f)				
A_L	B_L	S_L	A_R	B_R	S_R	A_R	B_R	S_R	A_R	B_R	S_R	A_L	B	A_R	S_L		$n_L + 1$	$n_R^1 + 1$	$n_R^2 + 1$	$n_R^3 + 1$
a_L^1	b^1	1.0	a_R^1	b^1	0.8	a_R^5	b^2	1.0	a_R^9	b^3	0.5	a_L^1	b^1	a_R^2	1.8	\tilde{S}	0.2	0.2	0.6	0.5
a_L^2	b^2	0.9	a_R^2	b^1	0.5	a_R^6	b^2	0.8	a_R^{11}	b^3	0.4	a_L^2	b^1	a_R^3	1.5	$\hat{\tau}$	1.6	1.6	1.5	1.6
a_L^3	b^2	0.8	a_R^3	b^1	0.2	a_R^7	b^2	0.7	a_R^{12}	b^3	0.4	a_L^3	b^2	a_R^4	1.9	$\Delta\mathcal{K}_{any}$	$\frac{1}{4}2 + \frac{1}{2}3 + \frac{1}{4}1$	1	2	1
a_L^4	b^3	0.5	a_R^4	b^1	0.1	a_R^8	b^2	0.6	a_R^{13}	b^3	0.3	a_L^5	b^2	a_R^5	1.7	$\Delta\mathcal{K}_{top,a}$	0	0	2	0
a_L^5	b^3	0.2										a_L^6	b^2	a_R^6	1.6	$\Delta\mathcal{K}_{top,b}$	$\frac{1}{4}0 + \frac{1}{2}0 + \frac{1}{4}0$	0	1	0
a_L^6	b^2	0.1										a_L^7	b^2	a_R^7	1.8	\mathcal{J}	$\frac{1}{8}$	$\frac{1}{4}$	$\frac{3}{4}$	$\frac{1}{4}$
												a_L^8	b^2	a_R^8	1.6	p	0.5385	0.0659	0.3297	0.0659
												a_L^9	b^2	a_R^9	1.5					
												a_L^{10}	b^3	a_R^{10}	1.0					

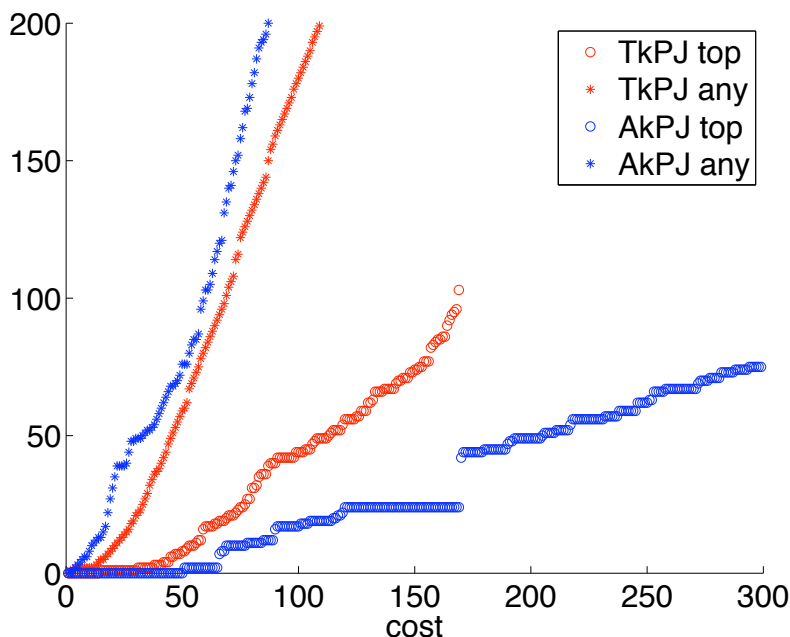


Fig. 1. Any-k (in blue) vs. top-k (in red) execution strategy. Stars (*) denote the number of *any-combinations*, while circles (o) the number of *top-combinations*.

the number of tuples per join attribute value varies, and it is sampled from a Poisson distribution with average $Q_i = 40$. Scores are generated as samples of an exponential distribution. In order to create non-homogeneous scores between the two services, we set the average of the exponential distribution to be, resp., 1 and 10. We assume equal access costs for the two services, so that AkPJ behaves as naively as possible, since it cannot differentiate the services by their costs.

Figure 1 illustrates the results obtained for one exemplary instance of the synthetic dataset (the y-axis indicates the number of combinations (either *top-combinations* or *any-combinations* and the x-axis the incurred cost). We notice that, in terms of *top-combinations*, TkPJ outperforms AkPJ by a wide margin, while the opposite holds as for *any-combinations*.

VI. CONCLUDING REMARKS

In this paper we proposed different execution strategies for pipe joins. We remark that the *AkPJ* strategy requires only the availability of estimated access costs. Conversely, the *TkPJ* strategy needs an estimate of the parameter J , i.e. number of

distinct values for B_L in s_L , in order to compute the threshold value. In practice this number can be estimated from cached query results.

Acknowledgments.: The authors acknowledge support from the ‘‘Search Computing’’ (SeCo) project, funded by the ERC under the 2008 Call for ‘‘IDEAS Advanced Grants’’. The authors also wish to thank the anonymous reviewers for their helpful comments on the paper.

REFERENCES

- [1] I. F. Ilyas, W. G. Aref, and A. K. Elmagarmid, ‘‘Supporting top-k join queries in relational databases,’’ in *VLDB*, 2003, pp. 754–765.
- [2] I. F. Ilyas, G. Beskales, and M. A. Soliman, ‘‘A survey of top- query processing techniques in relational database systems,’’ *ACM Comput. Surv.*, vol. 40, no. 4, 2008.
- [3] K. Schnaitter and N. Polyzotis, ‘‘Evaluating rank joins with optimal cost,’’ in *PODS*, 2008, pp. 43–52.