Matthias Boehm, Dirk Habich, Wolfgang Lehner

Multi-flow Optimization via Horizontal Message Queue Partitioning

Erstveröffentlichung in / First published in:

Enterprise Information Systems: 12th International Conference. Funchal-Madeira, 08.-12.06.2010. Springer, S. 31-47. ISBN 978-3-642-19802-1.

DOI: https://doi.org/10.1007/978-3-642-19802-1_3

Final edited form was published in "Enterprise Information Systems: 12th International Conference. Funchal-Madeira 2010", S. 31-47, ISBN 978-3-642-19802-1
https://doi.org/10.1007/978-3-642-19802-1_3

Multi-flow Optimization via Horizontal Message Queue Partitioning

Matthias Boehm, Dirk Habich, and Wolfgang Lehner

Dresden University of Technology, Database Technology Group, Dresden, Germany
{matthias.boehm,dirk.habich,wolfgang.lehner}@tu-dresden.de

Abstract. Integration flows are increasingly used to specify and execute data-intensive integration tasks between heterogeneous systems and applications. There are many different application areas such as near real-time ETL and data synchronization between operational systems. For the reasons of an increasing amount of data, highly distributed IT infrastructures, as well as high requirements for up-to-dateness of analytical query results and data consistency, many instances of integration flows are executed over time. Due to this high load, the performance of the central integration platform is crucial for an IT infrastructure. With the aim of throughput maximization, we propose the concept of multi-flow optimization (MFO). In this approach, messages are collected during a waiting time and executed in batches to optimize sequences of plan instances of a single integration flow. We introduce a horizontal (value-based) partitioning approach for message batch creation and show how to compute the optimal waiting time. This approach significantly reduces the total execution time of a message sequence and hence, it maximizes the throughput, while accepting moderate latency time.

Keywords: Integration flows, Multi-flow Optimization, Horizontal partitioning, Message queues, Throughput improvement.

1 Introduction

The scope of data management is continuously changing from the management of locally stored data towards the management of distributed information across multiple heterogeneous applications and systems. In this context, typically, integration flows are used in order to specify and execute complex procedural integration tasks. These integration flows are executed by message-oriented integration platforms such as EAI servers (Enterprise Application Integration) or MOM systems (Message-Oriented Middleware). For two reasons, many independent instances of an integration flow are executed over time. First, there is the requirement of immediate data synchronization between operational source systems in order to ensure data consistency. Second, data changes of the operational source systems are directly propagated into the data warehouse infrastructure in order to achieve high up-to-dateness of analytical query results (near real-time ETL). Due to this high load of flow instances, the performance of the central integration platform is crucial. Thus, optimization is required.

In the context of integration platforms, especially, in scenarios with high load of plan instances, the major optimization objective is throughput maximization [1] rather than

Final edited form was published in "Enterprise Information Systems: 12th International Conference. Funchal-Madeira 2010", S. 31-47, ISBN 978-3-642-19802-1
https://doi.org/10.1007/978-3-642-19802-1_3

the execution time minimization of single plan instances. Thus, the goal is to maximize the number of messages processed per time period. Here, moderate latency times of single messages are acceptable [2]. When optimizing integration flows, the following problems have to be considered:

Problem 1. Expensive External System Access. The time-expensive access of external systems is caused by network latency, network traffic, and message transformations into internal representations. The fact that external systems are accessed with similar queries over time offers potential for further optimization.

Problem 2. Cache Coherency Problem. One solution to Problem 1 might be the caching of results of external queries. However, this fails, because when integrating highly distributed systems and applications (loosely coupled without any notification mechanisms), the central integration platform cannot ensure that the cached data is consistent with the data in the source systems [1].

Problem 3. Serialized External Behavior. In dependence on the involved external systems, we need to ensure the serial order of messages. For example, this can be caused by referential integrity constraints within the target systems. Thus, we need to guarantee monotonic reads and writes for individual data objects.

Given these problems, the optimization objective of throughput maximization has so far only been addressed by leveraging a higher degree of parallelism, such as (1) intra-operator (horizontal) parallelism (data partitioning, see [3]), (2) inter-operator (horizontal) parallelism (explicit parallel subflows, see [4,9]), and (3) inter-operator (vertical) parallelism (pipelining of messages, see [5,6,7]). Although these techniques can increase the resource utilization and thus, increase the throughput, they do not reduce the executed work.

In this paper, we introduce the concept of *multi-flow optimization (MFO)* [8] in order to maximize the message throughput by reducing the executed work. Therefore, we periodically collect incoming messages and execute whole message batches with single plan instances. The novel idea is to use horizontal (value-based) message queue partitioning as a batch creation strategy and to compute the optimal waiting time. There, all messages of one batch (partition) exhibit the same attribute value with regard to a chosen partitioning attribute. This yields throughput improvements due to operation execution on partitions instead of on individual messages. In detail, we make the following contributions:

- First of all, in Section 2, we present an architecture of an integration platform and we give a solution overview of MFO via horizontal partitioning.
- In Section 3, we introduce the concept of a partition tree and we discuss the derivation of partitioning schemes as well as the related rewriting of plans.
- Then, in Section 4, we define the formal MFO problem. Here, we also explain the cost estimation and the computation of the optimal waiting time.
- Afterwards, we illustrate the results of our evaluation in Section 5.

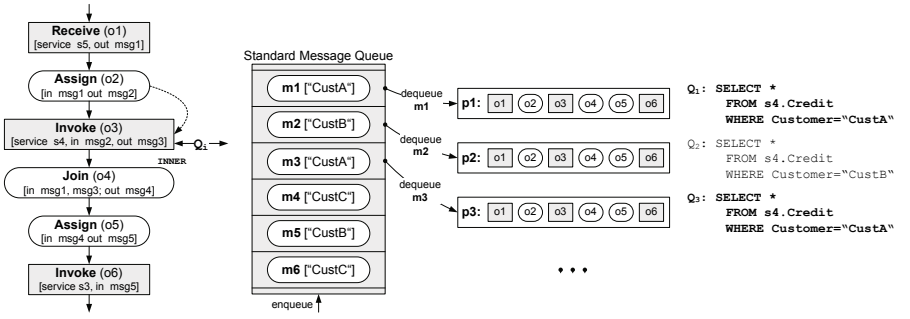
Finally, we analyze related work in Section 6 and conclude the paper in Section 7.

Final edited form was published in "Enterprise Information Systems: 12th International Conference. Funchal-Madeira 2010", S. 31-47, ISBN 978-3-642-19802-1
https://doi.org/10.1007/978-3-642-19802-1_3

2 System Architecture and Solution Overview

A typical integration platform system architecture consists of a set of inbound adapters, multiple message queues, an internal scheduler, a central process execution engine, and a set of outbound adapters. The inbound adapters passively listen for incoming messages, transform them into a common format (e.g., XML) and append the messages to message queues or directly forward them to the process engine. Within the process engine, compiled plans of deployed integration flows are executed. While executing those plans, the outbound adapters are used as services in order to actively invoke external systems. They transform the internal format back into the proprietary message representations. This architecture is also representative for major products such as SAP Process Integration, IBM Message Broker or MS Biztalk Server. The following example explains the instance-based (step-by-step) plan execution within such an architecture.

Example 1. Instance-Based Orders Processing: Assume a plan P_2 (Figure 1(a)). In the instance-based case, a new plan instance p_i is created for each incoming message (Figure 1(b)) and message queues are used at the inbound side only.



(a) Example Plan P_2 (b) Instance-Based Plan Execution of P_2

Fig. 1. Example Instance-Based Plan Execution

The `Receive` operator (o_1) gets an *orders* message from the queue and writes it to a local variable. Then, the `Assign` operator (o_2) prepares a query with the *customer name* of the received message as a parameter. Subsequently, the `Invoke` operator (o_3) queries the external system s_4 in order to load additional information for that customer. Here, one SQL query Q_i per plan instance (per message) is used. The `Join` operator (o_4) merges the result message with the received message (with the *customer key* as join predicate). Finally, the pair of `Assign` and `Invoke` operators (o_5 and o_6) sends the result to system s_3. We see that multiple orders from one customer (`CustA`: m_1, m_3) cause us to pose the same query (`Invoke` operator o_3) multiple times to the external system s_4.

Finally, we may end up with work done multiple times. At this point, multi-flow optimization comes into play, where we consider optimizing the whole sequence of plan instances. Our core idea is to periodically collect incoming messages and to execute whole message batches with single plan instances.

Final edited form was published in "Enterprise Information Systems: 12th International Conference. Funchal-Madeira 2010", S. 31-47, ISBN 978-3-642-19802-1
https://doi.org/10.1007/978-3-642-19802-1_3

Batch Creation via Horizontal Queue Partitioning

The naïve (time-based) batching approach, as already proposed for distributed queries [1] and web service interactions [9], is to collect messages during a waiting time Δtw, merge those messages to message batches b_i, and then execute a plan instance p'_i for each batch. Due to the time-based model of collecting messages, there might be multiple distinct messages in the batch according to certain operator predicates. Hence, we need to rewrite the queries to external systems, which (1) might not be possible for certain applications, and which (2) possibly negative performance influence cannot be precisely estimated due to the loose coupling of involved systems [10]. To tackle these problems, we propose a novel concept of MFO via horizontal message queue partitioning.

The basic idea is to horizontally partition the inbound message queues according to specific partitioning attributes ba. With such a value-based partitioning, all messages of a batch exhibit the same attribute value according to the partitioning attribute. Thus, several operators of the plan only need to access this attribute once for the whole partition rather than for each individual message. The core steps are (1) to derive partitioning attributes from the integration flow, (2) to periodically collect messages during a waiting time Δtw, (3) to read the first partition from the queue and (4) to execute the messages of this partition as a batch with a single plan instance. Additionally, (5) we might need to ensure the serial order of messages at the outbound side.

Example 2. Partitioned Batch-Orders Processing: Figure 2 reconsiders our example for partitioned multi-flow execution.

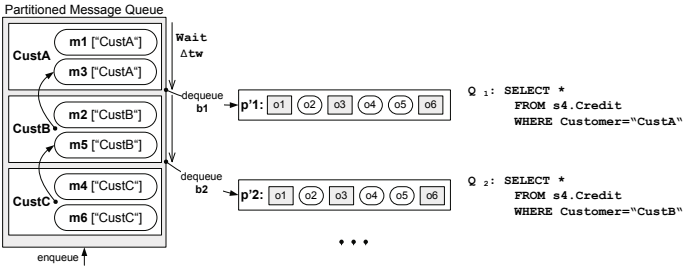


Fig. 2. Partitioned Message Batch Execution P'_2

The incoming messages m_i are partitioned according to the partitioning attribute *customer name* that was extracted with $ba = m_i/Customer/Cname$ at the inbound side. A plan instance of the rewritten plan P'_2 reads the first partition from the queue and executes the whole partition. Due to the equal values of the partitioning attribute, we do not need to rewrite the query to the external system s_2. Every batch contains one distinct attribute value according to ba. We achieve performance benefits for the `Assign`, as well as the `Invoke` operators.

It is important to note that beside external queries (`Invoke` operator), also local operators (e.g., `Assign` and `Switch`), but also operators that work on externally loaded data, can directly benefit from horizontal partitioning. This benefit is caused by operation execution on partitions instead of on individual messages.

Final edited form was published in "Enterprise Information Systems: 12th International Conference. Funchal-Madeira 2010", S. 31-47, ISBN 978-3-642-19802-1
https://doi.org/10.1007/978-3-642-19802-1_3

Clearly, MQO (Multi-Query Optimization) and OOP (Out-of-Order Processing) [11] have already been investigated for other system types. In contrast to existing work, we present the novel MFO approach that maximizes the throughput by computing the optimal waiting time. MFO is also related to caching and the recycling of intermediate results [12]. While caching might lead to using outdated data, the partitioned execution might cause reading more recent data. However, we cannot ensure strong consistency by using asynchronous integration flows (decoupled from clients with message queues) anyway. Further, we guarantee (1) monotonic writes, (2) monotonic reads with regard to individual data objects, (3) that the temporal gap is at most equal to a given latency constraint and (4) that no outdated data is read. In conclusion, caching is advantageous if data of external sources is static and the amount of data is rather small, while MFO is beneficial if data of external sources changes dynamically.

The major research challenges of MFO via horizontal partitioning are (1) to enable plan execution of message partitions and (2) to periodically compute the optimal waiting time Δtw. Both are addressed in the following sections.

3 Horizontal Queue Partitioning

In order to enable plan execution of message partitions, several preconditions are required. In this section, we describe (1) the horizontally partitioned message queue data structure *partition tree*, (2) the automatic derivation of the optimal partitioning scheme of such a tree, and (3) the related rewriting of plans.

3.1 Maintaining Partition Trees

As the foundation for multi-flow optimization, we introduce the *partition tree* as a partitioned message queue data structure. This partition tree is an extended multi-dimensional B*-Tree (MDB-Tree) [13], where the messages are horizontally (value-based) partitioned according to multiple attributes. Similar to a traditional MDB-Tree, each tree level represents a different partitioning attribute. In contrast, due to the queuing semantics, the major difference to traditional MDB-Trees is that the partitions are sorted according to their timestamps of creation rather than according to their key values. Thus, at each tree level, a list of partitions, unsorted with regard to the attribute values, is stored:

Definition 1. Partition Tree*: The partition tree is an index for multi-dimensional attributes. It contains h levels, where each level represents a partition attribute $ba_i \in \{ba_1, ba_2, \ldots, ba_h\}$. For each attribute ba_i, a list of batches (partitions) b are maintained. Those partitions are ordered according to their timestamps of creation $t_c(b_i)$ with $t_c(b_{i-1}) \leq t_c(b_i) \leq t_c(b_{i+1})$. The last index level ba_h contains the messages. A partition attribute has a $type(ba_i) \in \{\texttt{value}, \texttt{value-list}, \texttt{range}\}$.*

Such a partition tree is used as our message queue that, similar to normal message queues, decouples the inbound adapters from the process engine to handle overload situations but additionally, realizes the horizontal partitioning.

Final edited form was published in "Enterprise Information Systems: 12th International Conference. Funchal-Madeira 2010", S. 31-47, ISBN 978-3-642-19802-1
https://doi.org/10.1007/978-3-642-19802-1_3

Example 3. Partition Tree with $h = 2$: Assume two partitioning attributes ba_1 (customer, `value`) and ba_2 (total price, `range`) that have been derived from a plan P. Then, the partition tree exhibits a height of $h = 2$ (Figure 3).

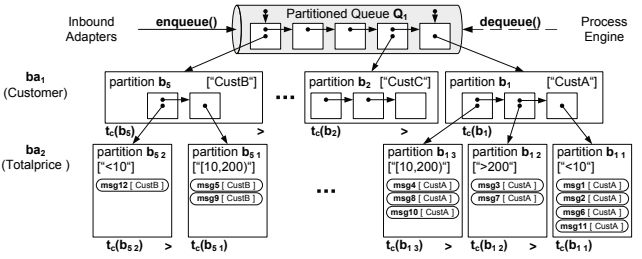


Fig. 3. Example Partition Tree

On the first index level, the messages are partitioned according to customer names $ba_1(m_i)$, and on the second level, each partition is divided according to the range of order total prices $ba_2(m_i)$. Horizontal partitioning in combination with the temporal order of partitions reason the outrun of single messages, while the messages within a partition are still sorted according to their incoming order.

Such a partition tree message queue has two operations: The `enqueue` operation is invoked by the inbound adapters whenever a message was received and transformed into the internal representation. During this message transformation, the values of registered partitioning attributes are extracted as well. We use a thread monitor approach for synchronization of enqueue and dequeue operations. Subsequently, we iterate over the list of partitions from the back to the front and determine whether or not a partition with $ba_l(m_i) = ba(b_i)$ already exists. If so, the message is inserted; otherwise, a new partition is created and added at the end of the list. In case of a node partition, we recursively invoke our algorithm, while in case of a leaf partition, the message is added to the end of the list. Due to linear comparison, the `enqueue` operation exhibits a worst-case time complexity of $O(\sum_i^h 1/sel(ba_i))$, where $sel(ba_i)$ denotes the average selectivity of a single partitioning attribute. In contrast, the `dequeue` operation is invoked by the process engine according to the computed waiting time Δtw. It simply removes and returns the first partition with $b^- \leftarrow b_1$ from the list of partitions at index level 1 with a constant worst-case time complexity of $O(1)$. This partition exhibits the property of being the oldest partition within the partition tree with $(\min_{i=1}^{|b|} t_c(b_i))$. This property ensures that starvation of messages is impossible.

With regard to robustness, we extend the partition tree to the *hash partition tree* that is a partition tree with a hash table as a secondary index over the partitioning attribute. This reduces the complexity for `enqueue` operations—in case no serialized external behavior (SEB) is required—to constant time of $O(1)$.

Final edited form was published in "Enterprise Information Systems: 12th International Conference. Funchal-Madeira 2010", S. 31-47, ISBN 978-3-642-19802-1
https://doi.org/10.1007/978-3-642-19802-1_3

3.2 Deriving Partitioning Schemes

The partitioning scheme in terms of a partition tree layout is derived automatically from a given plan. This includes (1) deriving candidate partitioning attributes and (2) to find the optimal partitioning scheme for the partition tree.

The candidate partition attributes are derived from the operators o_i of plan P. This includes the linear search with $O(m)$ for attributes that are involved in predicates, expressions and dynamic parameter assignments. We distinguish between the three partitioning attribute types *value*, *value-list*, and *range*. After having derived the set of candidate attributes, we select candidates that are advantageous to use. First, we remove all candidates, where a partitioning attribute refers to externally loaded data because these attribute values are not present at the inbound side. Second, we remove candidates, which benefit does not exceed a user-specified cost reduction threshold τ regarding the plan costs.

Based on the set of partitioning attributes, we can create a concrete partitioning scheme for a partition tree with regard to a single plan. For h partitioning attributes, there are $h!$ different partitioning schemes. The intuition of our heuristic of finding the optimal scheme is to minimize the number of partitions in the index. Therefore, we order the index attributes according to their selectivities with $\min \sum_{i=1}^{h} |b \in ba_i|$ iff $sel(ba_1) \geq sel(ba_i) \geq sel(ba_h)$ and thus, with complexity of $O(h \log h)$. Having minimized the total number of partitions, we minimized the overhead of queue maintenance and maximized the number of messages per top-level partition, which results in highest message throughput.

3.3 Plan Rewriting Algorithm

With regard to executing message partitions, only slight changes on physical level are required. First, the message meta model is extended such that an abstract message can be implemented by an atomic message or a message partition. Second, all operators that benefit from partitioning are modified accordingly. All other changes are made on logical level when rewriting a plan P to P' during the initial deployment or during periodical re-optimization.

For the purpose of plan rewriting, the flow meta model is extended by two additional operators: `PSlit` and `PMerge`. Then, the logical plan rewriting is realized with the so-called *split and merge* approach. From a macroscopic view, a plan receives the top-level partition, dequeued from the partition tree. Then, we can execute all operators that benefit from the top-level attribute. Just before an operator that benefits from a lower-level partition attribute, we need to insert a `PSplit` operator that splits the top-level partition into the $1/sel(ba_2)$ subpartitions (worst case) as well as an `Iteration` operator (foreach) that iterates over these subpartitions. The sequence of operators that benefit from this granularity are used as iteration body. After this iteration, we insert a `PMerge` operator in order to merge the resulting partitions back to the top-level partition if required. If we have only one partition attribute, we do not need to apply *split and merge*.

According to the requirement of serialized external behavior, we might need to serialize messages at the outbound side. Therefore, we extended the message by a counter c. If a message m_i outruns another message during `enqueue`, its counter $c(m_i)$

Final edited form was published in "Enterprise Information Systems: 12th International Conference. Funchal-Madeira 2010", S. 31-47, ISBN 978-3-642-19802-1
https://doi.org/10.1007/978-3-642-19802-1_3

is increased by one. Serialization is realized by timestamp comparison, and for each reordered message, the counter is decreased by one. Thus, at the outbound side, we are not allowed to send message m_i until $c(m_i) = 0$. It can be shown that the maximum latency constraints are still guaranteed [8].

4 Periodical Re-optimization

Apart from these prerequisites, the multi-flow optimization now reduces to computing the optimal waiting time for collecting messages. This is done by periodical cost-based optimization [14], where we estimate the costs and compute this waiting time with regard to minimizing the total latency time.

4.1 Formal Problem Definition

We assume a sequence of incoming messages $M = \{m_1, m_2, \ldots, m_n\}$, where each message m_i is modeled as a (t_i, d_i, a_i)-tuple, where $t_i \in \mathbb{Z}^+$ denotes the incoming timestamp of the message, d_i denotes a semi-structured tree of name-value data elements, and a_i denotes a list of additional atomic name-value attributes. Each message m_i is processed by an instance p_i of a plan P, and $t_{out}(m_i) \in \mathbb{Z}^+$ denotes the timestamp when the message has been executed. The latency of a single message $T_L(m_i)$ is given by $T_L(m_i) = t_{out}(m_i) - t_i(m_i)$. Furthermore, the execution characteristics of a finite message subsequence M' with $M' \subseteq M$ are described by two statistics. First, the total execution time $W(M')$ of a subsequence is determined by $W(M') = \sum W(p')$ as the sum of execution times of all partitioned plan instances required to execute M'. Second, the total latency time $T_L(M')$ of a subsequence is determined by $T_L(M') = t_{out}(m_{|M'|}) - t_i(m_1)$ as the time between receiving the first message until the last message has been executed. This includes overlapping execution time and waiting time.

Definition 2. Multi-Flow Optimization Problem (P-MFO): *Maximize the message throughput with regard to a finite message subsequence M'. The optimization objective ϕ is to execute M' with minimal latency time:*

$$\phi = \max \frac{|M'|}{\Delta t} = \min T_L(M'). \tag{1}$$

There, two additional restrictions must hold:

1. *Let lc denote a soft latency constraint that must not be exceeded significantly. Then, the condition $\forall m_i \in M' : T_L(m_i) \leq lc$ must hold.*
2. *The external behavior must be serialized according to the incoming message order. Thus, the condition $\forall m_i \in M' : t_{out}(m_i) \leq t_{out}(m_{i+1})$ must hold.*

Finally, the P-MFO describes the search for the optimal waiting time Δtw with regard to ϕ and the given constraints.

Based on the horizontal partitioning, Figure 4 illustrates the temporal aspects. An instance p'_i of a rewritten plan P' is initiated periodically at T_i, where the period is determined by the waiting time Δtw. Then, a message partition b_i is executed by p'_i with

Final edited form was published in "Enterprise Information Systems: 12th International Conference. Funchal-Madeira 2010", S. 31-47, ISBN 978-3-642-19802-1
https://doi.org/10.1007/978-3-642-19802-1_3

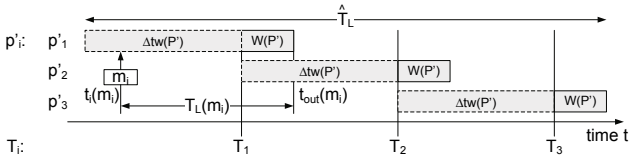


Fig. 4. P-MFO Temporal Aspects (with $\Delta tw > W(P')$)

an execution time of $W(P')$. Finally, we estimate the total latency time $\hat{T}_L(M')$ in order to solve the defined optimization problem. In order to overcome the problems of (1) temporally overlapping plan instances for different partitions, (2) growing message queue sizes (in case of $\Delta tw < W(P')$), and (3) any a-priori violated latency constraint, we define the following *validity condition*: For a given latency constraint lc, there must exist a waiting time Δtw such that $(0 \leq W(P') \leq \Delta tw) \wedge (0 \leq \hat{T}_L(|M'|) \leq lc)$; otherwise, the constraint is invalid. In other words, (1) we avoid the case $\Delta tw < W(P')$, and (2) we check if the worst-case message latency—in the sense of the total latency of $|M'| = 1/sel$ distinct message partitions—is lower than or equal to the latency constraint lc.

4.2 Extended Cost Model and Cost Estimation

In order to estimate the costs of plan instances for specific batch sizes k' with $k' = |b_i|$, we need to extend our cost model for integration flows with regard to these partitions. The extended costs $C(o'_i, k')$ of operators that benefit from partitioning (e.g., `Invoke`, `Assign`, and `Switch`) are independent of the number of messages k'. In contrast, the costs of all operators that do not benefit from partitioning linearly depend on the number of messages in the batch k'.

For operators that do not benefit from partitioning, the abstract costs are computed by $C(o'_i, k') = C(o_i) \cdot k'$ and the execution time can be computed by $W(o'_i, k') = W(o_i) \cdot k'$ or by $W(o'_i, k') = W(o_i) \cdot C(o'_i, k')/C(o_i)$. Finally, if $k' = 1$, we get the instance-based costs with $C(o'_i, k') = C(o_i)$ and $W(o'_i, k') = W(o_i)$. Thus, the instance-based execution is a specific case of horizontal partitioning (one message per batch). Using the extended cost model, we now can compute the total execution time $W(M', k')$ and the total latency time $T_L(M', k')$ of message subsequences M' by assuming $\lceil |M'|/k' \rceil = 1/sel$ instances of a partitioned plan. The estimated total execution time $\hat{W}(M')$ of a subsequence is computed by the estimated costs per instance times the number of executed plan instances:

$$\hat{W}(M', k') = \hat{W}(P', k') \cdot \left\lceil \frac{|M'|}{k'} \right\rceil \quad \text{with} \quad \hat{W}(P', k') = \sum_{i=1}^{m} \hat{W}(o', k'). \tag{2}$$

In contrast, the estimated total latency time $\hat{T}_L(M')$ of a message subsequence is composed of the waiting time Δtw and the execution time $\hat{W}(P', k')$. Thus, we compute it based on the comparison between Δtw and $\hat{W}(P', k')$ with

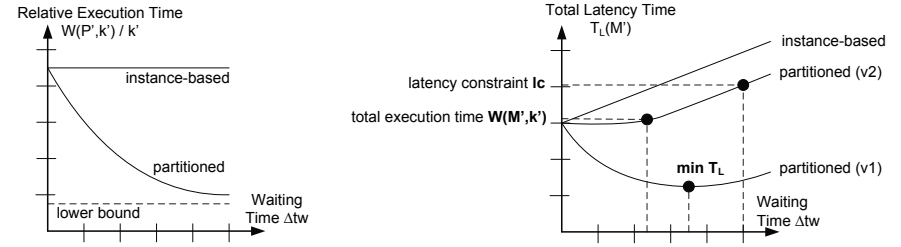
Final edited form was published in "Enterprise Information Systems: 12th International Conference. Funchal-Madeira 2010", S. 31-47, ISBN 978-3-642-19802-1
https://doi.org/10.1007/978-3-642-19802-1_3

$$\hat{T}_L(M',k') = \begin{cases} \left\lceil \frac{|M'|}{k'} \right\rceil \cdot \Delta tw + \hat{W}(P',k') & \Delta tw \geq W(P',k') \\ \Delta tw + \left\lceil \frac{|M'|}{k'} \right\rceil \cdot \hat{W}(P',k') & \text{otherwise.} \end{cases} \quad (3)$$

Due to our *validity condition*, $\Delta tw < W(P',k')$ is invalid and therefore we use only the first case of Equation 3. As a result, it follows that $\hat{W}(M',k') \leq \hat{T}_L(M',k')$, where $\hat{W}(M',k') = \hat{T}_L(M',k')$ is the case at $\Delta tw = W(P',k')$. Hence, the total latency time cannot be lower than the total execution time.

4.3 Waiting Time Computation

The intuition of computing the optimal waiting time Δtw with regard to minimizing the total latency time is that the waiting time—and hence, the batch size k'—strongly influences the execution time of single plan instances. Then, the latency time depends on that execution time. Figure 5 illustrates the resulting two inverse influences that our computation algorithm exploits:



(a) $\Delta tw \rightarrow W(P')/k'$ Influence (b) $\Delta tw \rightarrow T_L(M')$ Influence

Fig. 5. Search Space for Waiting Time Computation

As shown in Figure 5(a), for partitioned plan execution, an increasing waiting time Δtw causes decreasing relative execution time $W(P',k')/k'$ and total execution time $W(M',k')$, which both are non-linear functions that asymptotically tend towards a lower bound. In contrast, Δtw has no influence on instance-based execution times. Furthermore, Δtw also influences the latency time (see Figure 5(b)). On the one hand, an increasing waiting time Δtw linearly increases the latency time $\hat{T}_L$ because the waiting time is included in $\hat{T}_L$. On the other hand, an increasing Δtw causes a decreasing execution time and thus, indirectly decreases $\hat{T}_L$ because the execution time is included in $\hat{T}_L$. The result, in the general case of arbitrary cost functions, is a non-linear total latency time that has a local minimum (v1) or not (v2). In any case, due to the validity condition, the total latency function is defined for the closed interval $T_L(M',k') \in [W(M',k'), lc]$ and hence, both global minimum and maximum exist.

In order to compute the optimal waiting time, we monitor the incoming message rate $R \in \mathbb{R}$ and the value selectivity $sel \in (0,1]$ according to the partitioning attributes. The first partition will contain $k' = R \cdot sel \cdot \Delta tw$ messages. For the i-th partition

Final edited form was published in "Enterprise Information Systems: 12th International Conference. Funchal-Madeira 2010", S. 31-47, ISBN 978-3-642-19802-1
https://doi.org/10.1007/978-3-642-19802-1_3

with $i \geq 1/sel$, k' is computed by $k' = R \cdot \Delta tw$, independently of the selectivity sel. A low selectivity implies many partitions b_i but only few messages in a partition ($|b_i| = R \cdot sel$). However, the high number of partitions b_i forces us to wait longer ($\Delta tw/sel$) until the execution of a partition.

Based on the relationship between the waiting time Δtw and the number of message per batch k', we can compute the waiting time, where $\hat{T}_L$ is minimal by

$$\Delta tw \quad | \quad \min \hat{T}_L(\Delta tw) \quad \text{and} \quad \hat{W}(M', \Delta tw) \leq \hat{T}_L(M', \Delta tw) \leq lc \tag{4}$$

Using Equation 3 and assuming a fixed message rate R with $1/sel$ distinct items according to the partitioning attribute, we substitute k' with $R \cdot \Delta tw$ and get

$$\hat{T}_L(M', R \cdot \Delta tw) = \left\lceil \frac{|M'|}{R \cdot \Delta tw} \right\rceil \cdot \Delta tw + W(P', R \cdot \Delta tw) \tag{5}$$

Finally, we set $M' = k'/sel = (R \cdot \Delta tw)/sel$ (execution of all $1/sel$ distinct partitions, where each partition contains k' messages), use Equation 5 and compute

$$\begin{aligned} \Delta tw \quad | \quad \min \hat{T}_L(M', R \cdot \Delta tw) \quad \text{with} \quad & \hat{T}'_L(M', R \cdot \Delta tw)_{\Delta tw} = 0 \\ & \hat{T}''_L(M', R \cdot \Delta tw)_{\Delta tw \Delta tw} > 0. \end{aligned} \tag{6}$$

If such a local minimum exists, we check the validity of $(0 \leq W(P', k') \leq \Delta tw) \wedge (0 \leq \hat{T}_L \leq lc)$. If $\Delta tw < W(P', k')$, we search for $\Delta tw = W(P', k')$. Further, if $\hat{T}_L > lc$, we search for the Δtw with $\hat{T}_L(\Delta tw) = lc$. If such a minimum does not exist, we compute Δtw for the lower border of the interval with

$$\Delta tw \quad | \quad \min \hat{T}_L(M', R \cdot \Delta tw) \quad \text{with} \quad \hat{T}_L(M', R, \Delta tw) = W(M', \Delta tw \cdot R). \tag{7}$$

This lower border of $W(M', k')$ is given at $\Delta tw = W(P', k')$, where $W(P', k')$ depends itself on Δtw. With regard to the load situation, there might not be a valid $\Delta tw = W(P', k')$ with $\Delta tw \geq 0$. In this overload situation, we compute the maximum number of messages per batch k'' by $k'' \mid W(M', k'') = T_L(M', k'') = lc$. In this case, the waiting time is $\Delta tw = W(P'', k'')$ and we have a full utilization $W(M', k'') = T_L(M', k'')$. However, we do not execute the partition with all collected messages k' but only with the k'' messages, while the $k' - k''$ messages are reassigned to the end of the partition tree (and we modify the outrun counters with regard to serialized external behavior). Thus, we achieve highest throughput but still can ensure the maximum latency constraint.

Our extended cost model includes only two categories of operator costs. Hence, we compute the waiting time using an tailor-made algorithm with complexity of $O(m)$. It computes the costs $W^-(P)$ that are independent of k' and the costs $W^+(P)$ that depend linearly on k'. Using a simplified Equation 7, we compute Δtw at the lower border of the defined T_L function interval with

$$\Delta tw = W(P', \Delta tw \cdot R) = W^-(P') + W^+(P') \cdot \Delta tw \cdot R = \frac{W^-(P')}{1 - W^+(P') \cdot R}. \tag{8}$$

Final edited form was published in "Enterprise Information Systems: 12th International Conference. Funchal-Madeira 2010", S. 31-47, ISBN 978-3-642-19802-1
https://doi.org/10.1007/978-3-642-19802-1_3

Finally, we check the validity condition in order to react on overload situations. As a result, we get the optimal waiting time that minimizes the total latency time and thus, maximizes the message throughput of a single deployed plan.

5 Experimental Evaluation

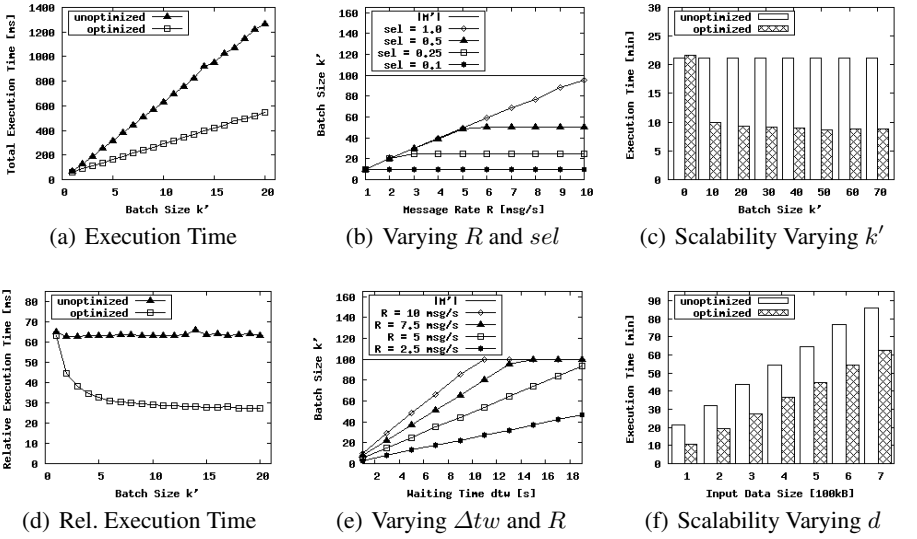
Our evaluation shows that (1) significant throughput improvements are reachable, (2) the maximum latency guarantees hold under experimental investigation, and (3) the runtime overhead is negligible. We implemented the approach of MFO via horizontal partitioning within our java-based `WFPE` (workflow process engine) and integrated it into our cost-based optimization framework. This includes the (hash) partition tree, slightly changed operators as well as the algorithms for deriving partitioning attributes (A-DPA), plan rewriting (A-MPR) and waiting time computation (A-WTC). Furthermore, we ran our experiments on an IBM blade (OS Suse Linux, 32bit) with two processors (each of them a Dual Core AMD Opteron Processor 270 at 2 GHz) and 9 GB RAM. We used synthetically generated datasets in order to simulate arbitrary selectivities and cardinalities. As the integration flow under test, we used our example plan P_2.

5.1 Execution Time and Scalability

First, we evaluated the execution time of partitioned plan execution compared to the unoptimized execution. We varied the batch size $k' \in [1, 20]$, measured the execution time $W(P_2', k')$ (Figure 6(a)) and computed the relative execution time $W(P_2', k')/k'$ (Figure 6(d)). For comparison, the unoptimized plan was executed k' times, where we measured the total execution time. This experiment has been repeated 100 times. The unoptimized execution shows a linear scalability with increasing batch size k', where the logical y-intercept is zero. In contrast, the optimized plan also shows a linear scalability but with a higher logically y-intercept. As a result, the relative execution time is constant for the unoptimized execution, while for the optimized execution it decreases with increasing batch size and tends towards a lower bound. This lower bound is given by the costs of operators that do not benefit from partitioning. Note that (1) even for one-message-partitions the overhead is negligible and (2) that even small numbers of messages within a batch significantly reduce the relative execution time.

Second, we investigated the inter-influences between message arrival rates R, waiting times Δtw, partitioning attribute selectivities sel and the resulting batch sizes k' ($k' = R \cdot \Delta tw$). We executed $M' = 100$ messages and repeated all subexperiments 100 times. As a first subexperiment, we fixed a waiting time of $\Delta tw = 10\,\text{s}$. Figure 6(b) shows the influence of the message rate R on the average number of messages in the batch. We can observe (1) that the higher the message rate, the higher the number of messages per batch, and (2) that the selectivity determines the reachable upper bound. However, the influence of the message rate is independent of the selectivity. As a second subexperiment, Figure 6(e) illustrates the influence of Δtw on the batch size k', where we fixed $sel = 1.0$. Both an increasing waiting time and an increasing message rate, increase the batch size until the total number of messages is reached.

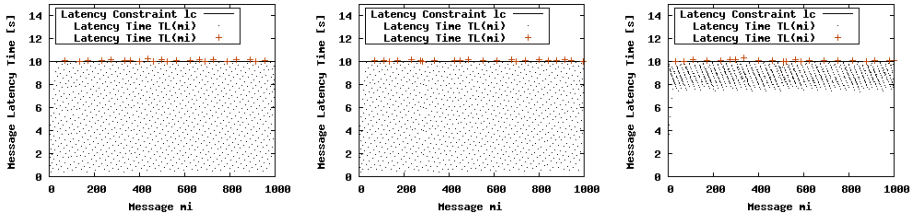
Final edited form was published in "Enterprise Information Systems: 12th International Conference. Funchal-Madeira 2010", S. 31-47, ISBN 978-3-642-19802-1
https://doi.org/10.1007/978-3-642-19802-1_3



(a) Execution Time (b) Varying R and sel (c) Scalability Varying k'

(d) Rel. Execution Time (e) Varying Δtw and R (f) Scalability Varying d

Fig. 6. Execution Time, Scalability and Influences on Batch Sizes

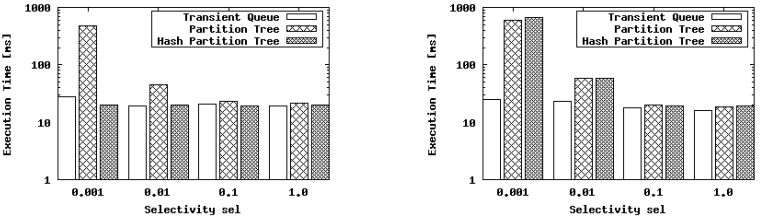


(a) Fixed w/o SEB (b) Poisson w/o SEB (c) Fixed w/ SEB

Fig. 7. Latency Time of Single Messages $T_L(m_i)$

Third, we investigate the scalability of plan execution. This includes the scalability (1) with increasing input data size, and (2) with increasing batch size. We executed 20,000 plan instances and compared the optimized plans with their unoptimized counterparts, where we fixed an optimization interval of $\Delta t = 5\,\text{min}$. In a first subexperiment, we investigated the scalability with increasing batch size $k' \in \{1, 10, 20, 30, 40, 50, 60, 70\}$. Figure 6(c) shows the results. We observe, that the overhead for executing one-message-partitions is marginal, which is reasoned by additional abstractions for messages and operators. Furthermore, we see the monotonically non-increasing total execution time function (with increasing batch size) and the existence of a lower bound of the total execution time. In a second subexperiment, we varied the input data size $d \in \{1, 2, 3, 4, 5, 6, 7\}$ (in 100 kB), while the size of externally loaded data was unchanged. We fixed a batch size of $k' = 10$. The results are shown

Final edited form was published in "Enterprise Information Systems: 12th International Conference. Funchal-Madeira 2010", S. 31-47, ISBN 978-3-642-19802-1
https://doi.org/10.1007/978-3-642-19802-1_3



(a) w/o Serialization (SEB) (b) w/ Serialization (SEB)

Fig. 8. Runtime Overhead for Enqueue with Different Message Queues

in Figure 6(f). In general, we observe good scalability with increasing data size. However, the relative improvement is decreasing because the size of loaded data was not changed. We can conclude that the scalability depends on the workload and on the concrete plan.

5.2 Latency Time

Furthermore, we executed $M' = 1{,}000$ messages using a maximum latency constraint of $lc = 10\,\mathrm{s}$ and measured the message latency times $T_L(m_i)$. We fixed a selectivity of $sel = 0.1$, a message arrival rate of $R = 5\,{}^{\mathrm{msg}}/_{\mathrm{s}}$ and used different arrival rate distributions (fixed, poisson) as well as analyzed the influence of serialized external behavior (SEB). As a worst-case consideration, we computed the waiting time $\Delta tw \mid T_L(M' = k'/sel) = lc$, which resulted in $\Delta tw = 981.26\,\mathrm{ms}$ because in the worst case, there are $1/sel = 10$ different partitions plus the execution time of the last partition. For both message arrival rate distribution functions (Figure 7(a) and Figure 7(b)), the constraint is not significantly exceeded. The latency of messages varies from almost zero to the latency constraint, where the missed constraints are caused by variations of the execution time. The constraint also holds for SEB, where all messages show more similar latency times (Figure 7(c)) due to serialization at the outbound side.

5.3 Runtime Overhead

We analyzed the overhead of the (hash) partition tree compared to the transient message queue. We enqueued 20,000 messages (see scalability experiments) with varying selectivities $sel \in \{0.001, 0.01, 0.1, 1.0\}$ and measured the total execution time. This experiment was repeated 100 times. Figure 8 illustrates the results using log-scaled x- and y-axes. We see that without SEB, both the transient queue and the hash partition tree show constant execution time and the overhead of the hash partition tree is negligible. In contrast, the execution time of the partition tree linearly increases with decreasing selectivity due to the linear probing over all partitions. We observe a similar behavior with SEB, except that the execution time of the hash partition tree also increases linearly with decreasing selectivity. This is caused by the required counting of outrun messages.

Final edited form was published in "Enterprise Information Systems: 12th International Conference. Funchal-Madeira 2010", S. 31-47, ISBN 978-3-642-19802-1
https://doi.org/10.1007/978-3-642-19802-1_3

In conclusion, MFO leads to throughput improvements by accepting moderate additional latency time. How much we benefit depends on the used plans and on the workload. The benefit is caused by (1) a moderate runtime overhead, and (2) the need for only few messages in a partition to yield a significant speedup.

6 Related Work

Multi-Query Optimization. The basic concepts of Multi-Query Optimization (MQO) [15] are *pipelined execution* and *data sharing*, where a huge body of work exists for local environments [16,17] and for distributed query processing [10,18,1,17]. For example, Lee et al. employed the waiting opportunities within a blocking query execution plan [1]. Further, Qiao et al. investigated a batch-sharing partitioning scheme [19] in order to allow similar queries to share cache contents. The main difference is that MQO benefits from reusing results across queries, while for MFO, this is impossible due to incoming streams of disjoint messages. In addition, MFO computes the optimal waiting time.

Data Partitioning. Horizontal data partitioning [20] is strongly applied in DBMS and distributed systems. Typically, this is an issue of physical design [21]. However, there are more recent approaches such as table partitioning along foreign-key constraints [22]. In the area of data streams, data partitioning was used for plan partitioning across server nodes [23] or single filter evaluation on tuple granularity [24]. Finally, there are also similarities to partitioning in parallel DBMS. The major difference is that MFO handles infinite streams of messages.

Workflow Optimization. In addition, there are data-centric but rule-based approaches of optimizing BPEL processes [25] and ETL flows [26]. In contrast, we proposed the cost-based optimization of integration flows [14]. Anyway, these approaches focus on execution time minimization rather than on throughput maximization. Furthermore, there are existing approaches [5,6,4,9] that also address throughput optimization. However, those approaches increase the degree of parallelism, while our approach reduces executed work.

7 Conclusions

To summarize, we proposed a novel approach for throughput maximization of integration flows that reduces work by employing horizontal data partitioning. Our evaluation showed that significant performance improvements are possible and that theoretical latency guarantees also hold under experimental investigation. In conclusion, the MFO approach can seamlessly be applied in a variety of integration platforms that asynchronously execute data-driven integration flows.

The general MFO approach opens many opportunities for further optimizations. Future work might consider (1) the execution of partitions independent of their temporal order, (2) plan partitioning in the sense of compiling different plans for different partitions, (3) global MFO for multiple plans, and (4) the cost-based plan rewriting. Finally, we might (5) combine MFO with pipelining and load balancing because both address throughput maximization as well.

Final edited form was published in "Enterprise Information Systems: 12th International Conference. Funchal-Madeira 2010", S. 31-47, ISBN 978-3-642-19802-1
https://doi.org/10.1007/978-3-642-19802-1_3

References

1. Lee, R., Zhou, M., Liao, H.: Request window: an approach to improve throughput of rdbms-based data integration system by utilizing data sharing across concurrent distributed queries. In: VLDB (2007)
2. Cecchet, E., Candea, G., Ailamaki, A.: Middleware-based database replication: the gaps between theory and practice. In: SIGMOD (2008)
3. Bhide, M., Agarwal, M., Bar-Or, A., Padmanabhan, S., Mittapalli, S., Venkatachaliah, G.: Xpedia: Xml processing for data integration. PVLDB 2(2) (2009)
4. Li, H., Zhan, D.: Workflow timed critical path optimization. Nature and Science 3(2) (2005)
5. Biornstad, B., Pautasso, C., Alonso, G.: Control the flow: How to safely compose streaming services into business processes. In: SCC (2006)
6. Boehm, M., Habich, D., Preissler, S., Lehner, W., Wloka, U.: Cost-based vectorization of instance-based integration processes. In: Grundspenkis, J., Morzy, T., Vossen, G. (eds.) ADBIS 2009. LNCS, vol. 5739, pp. 253–269. Springer, Heidelberg (2009)
7. Preissler, S., Habich, D., Lehner, W.: Process-based data streaming in service-oriented environments. In: Filipe, J., Cordeiro, J. (eds.) ICEIS 2010. LNBIP, vol. 73, pp. 60–75. Springer, Heidelberg (2010)
8. Boehm, M., Habich, D., Lehner, W.: Multi-process optimization via horizontal message queue partitioning. In: Filipe, J., Cordeiro, J. (eds.) Graph Grammars 1978. LNBIP, vol. 73, pp. 31–47. Springer, Heidelberg (2010)
9. Srivastava, U., Munagala, K., Widom, J., Motwani, R.: Query optimization over web services. In: VLDB (2006)
10. Ives, Z.G., Halevy, A.Y., Weld, D.S.: Adapting to source properties in processing data integration queries. In: SIGMOD (2004)
11. Li, J., Tufte, K., Shkapenyuk, V., Papadimos, V., Johnson, T., Maier, D.: Out-of-order processing: a new architecture for high-performance stream systems. PVLDB 1(1) (2008)
12. Ivanova, M., Kersten, M.L., Nes, N.J., Goncalves, R.: An architecture for recycling intermediates in a column-store. In: SIGMOD (2009)
13. Scheuermann, P., Ouksel, A.M.: Multidimensional b-trees for associative searching in database systems. Inf. Syst. 7(2) (1982)
14. Boehm, M., Wloka, U., Habich, D., Lehner, W.: Workload-based optimization of integration processes. In: CIKM (2008)
15. Roy, P., Seshadri, S., Sudarshan, S., Bhobe, S.: Efficient and extensible algorithms for multi query optimization. In: SIGMOD (2000)
16. Harizopoulos, S., Shkapenyuk, V., Ailamaki, A.: Qpipe: A simultaneously pipelined relational query engine. In: SIGMOD (2005)
17. Unterbrunner, P., Giannikis, G., Alonso, G., Fauser, D., Kossmann, D.: Predictable performance for unpredictable workloads. PVLDB 2(1) (2009)
18. Kementsietsidis, A., Neven, F., de Craen, D.V., Vansummeren, S.: Scalable multi-query optimization for exploratory queries over federated scientific databases. In: VLDB (2008)
19. Qiao, L., Raman, V., Reiss, F., Haas, P.J., Lohman, G.M.: Main-memory scan sharing for multi-core cpus. PVLDB 1(1) (2008)
20. Ceri, S., Negri, M., Pelagatti, G.: Horizontal data partitioning in database design. In: SIGMOD (1982)
21. Agrawal, S., Narasayya, V.R., Yang, B.: Integrating vertical and horizontal partitioning into automated physical database design. In: SIGMOD (2004)

Final edited form was published in "Enterprise Information Systems: 12th International Conference. Funchal-Madeira 2010", S. 31-47, ISBN 978-3-642-19802-1
https://doi.org/10.1007/978-3-642-19802-1_3

22. Eadon, G., Chong, E.I., Shankar, S., Raghavan, A., Srinivasan, J., Das, S.: Supporting table partitioning by reference in oracle. In: SIGMOD (2008)
23. Johnson, T., Muthukrishnan, S.M., Shkapenyuk, V., Spatscheck, O.: Query-aware partitioning for monitoring massive network data streams. In: SIGMOD (2008)
24. Avnur, R., Hellerstein, J.M.: Eddies: Continuously adaptive query processing. In: SIGMOD (2000)
25. Vrhovnik, M., Schwarz, H., Suhre, O., Mitschang, B., Markl, V., Maier, A., Kraft, T.: An approach to optimize data processing in business processes. In: VLDB (2007)
26. Simitsis, A., Vassiliadis, P., Sellis, T.K.: Optimizing etl processes in data warehouses. In: ICDE (2005)