Matthias Boehm, Dirk Habich, Steffen Preissler, Wolfgang Lehner, Uwe Wloka

Cost-Based Vectorization of Instance-Based Integration Processes

Erstveröffentlichung in / First published in:

Advances in Databases and Information Systems: 13th East European Conference. Riga, 07.-10.09.2009. Springer, S. 253-269. ISBN 978-3-642-03973-7.

DOI: https://doi.org/10.1007/978-3-642-03973-7_19

Final edited form was published in "Advances in Databases and Information Systems: 13th East European Conference. Riga 2009", S. 253-269, ISBN 978-3-642-03973-7
https://doi.org/10.1007/978-3-642-03973-7_19

Cost-Based Vectorization of Instance-Based Integration Processes

Matthias Boehm[1], Dirk Habich[2], Steffen Preissler[2], Wolfgang Lehner[2], and Uwe Wloka[1]

[1] Dresden University of Applied Sciences, Database Group
{mboehm,wloka}@informatik.htw-dresden.de
[2] Dresden University of Technology, Database Technology Group
{dirk.habich,steffen.preissler,wolfgang.lehner}@tu-dresden.de

Abstract. The inefficiency of integration processes—as an abstraction of workflow-based integration tasks—is often reasoned by low resource utilization and significant waiting times for external systems. With the aim to overcome these problems, we proposed the concept of process vectorization. There, instance-based integration processes are transparently executed with the pipes-and-filters execution model. Here, the term vectorization is used in the sense of processing a sequence (vector) of messages by one standing process. Although it has been shown that process vectorization achieves a significant throughput improvement, this concept has two major drawbacks. First, the theoretical performance of a vectorized integration process mainly depends on the performance of the most cost-intensive operator. Second, the practical performance strongly depends on the number of available threads. In this paper, we present an advanced optimization approach that addresses the mentioned problems. Therefore, we generalize the vectorization problem and explain how to vectorize process plans in a cost-based manner. Due to the exponential complexity, we provide a heuristic computation approach and formally analyze its optimality. In conclusion of our evaluation, the message throughput can be significantly increased compared to both the instance-based execution as well as the rule-based process vectorization.

Keywords: Cost-Based Vectorization, Integration Processes, Throughput Optimization, Pipes and Filters, Instance-Based.

1 Introduction

Integration processes—as an abstraction of workflow-based integration tasks—are typically executed with the *instance-based execution model* [1]. Here, each incoming message conceptually initiates a new instance of the related integration process. Therefore, all messages are serialized according to their incoming order. This order is then used to execute single-threaded process plans. Example system categories for that execution model are EAI (enterprise application integration) servers, WfMS (workflow management systems) and WSMS (Web service management systems). Workflow-based integration platforms usually do not reach high resource utilization because of (1) the existence of single-threaded process instances in parallel processor architectures, (2)

Final edited form was published in "Advances in Databases and Information Systems: 13th East European Conference. Riga 2009", S. 253-269, ISBN 978-3-642-03973-7
https://doi.org/10.1007/978-3-642-03973-7_19

significant waiting times for external systems, and (3) IO bottlenecks (message persistence for recovery processing). Hence, the message throughput is not optimal and can be significantly optimized using a higher degree of parallelism.

Other system types use the so-called *pipes-and-filters* execution model, where each operator is conceptually executed as a single thread and each edge between two operators contains a message queue. In order to overcome the problem of low resource utilization, in [2], we introduced the vectorization of instance-based integration processes. This approach describes the transparent rewriting of integration processes from the instance-based execution model to the pipes-and-filters execution model. In that context, different problems such as the assurance of serialized execution and different data flow semantics were solved. We use the term vectorization in analogy to the area of computational engineering because in the pipes-and-filters model, each operator executes a sequence (vector) of messages.

Although full vectorization can significantly increase the resource utilization and hence improve the message throughput, the two following major shortcomings exist:

Problem 1. Work-Cycle Domination*: The work-cycle of a whole data-flow graph is dominated by the work-cycle of the most cost-intensive operator because all queues after this operator are empty while operators in front reach the maximum constraint of the queues. Hence, the theoretical performance mainly depends on that operator.*

Problem 2. Overload Resource Utilization*: The practical performance strongly depends on the number of available threads. For full vectorization, the number of required threads is determined by the number of operators. Hence, if the number of required threads is higher than the number of available threads, performance will degenerate.*

In order to overcome those two drawbacks, in this paper, we propose the cost-based vectorization of integration processes. The core idea is to group the m operators of a process plan into k execution groups, then execute not each operator but each execution group with a single thread and hence, reduce the number of required threads. This approach is a generalization of the specific cases of instance-based ($k = 1$) and vectorized ($k = m$) integration processes. Therefore, here, we make the following contributions that also reflect the structure of the paper:

- First, in Section 2, we revisit the vectorization of integration processes, explain requirements and problems, and define the integration process vectorization problem.
- Subsequently, in Section 3, we introduce the sophisticated cost-based optimization approach. This approach overcomes the problem of possible inefficiency by applying the simple rule-based rewriting techniques.
- Based on the details in Sections 2 and 3, we provide conceptual implementation details and the results of our exhaustive experimental evaluation in Section 4.

Finally, we analyze the related work from different perspectives in Section 5 and conclude the paper in Section 6.

Final edited form was published in "Advances in Databases and Information Systems: 13th East European Conference. Riga 2009", S. 253-269, ISBN 978-3-642-03973-7
https://doi.org/10.1007/978-3-642-03973-7_19

2 Process Plan Vectorization Revisited

In this section, we recall the core concepts from the process plan vectorization approach. Therefore, we explain assumptions and requirements, define the vectorization problem and give an overview of the rule-based vectorization approach.

2.1 Process Vectorization Overview

Figure 1 illustrates an integration platform architecture for instance-based integration processes. Here, the key characteristics are a set of inbound adapters (passive listeners), several message queues, a central process engine, and a set of outbound adapters (active services). The message queues are used as logical serialization elements within the asynchronous execution model. However, the synchronous as well as the asynchronous execution of process plans is supported. Further, the process engine is instance-based, which means that for each message, a new instance (one thread) of the specified process plan is created and executed. In contrast to traditional query optimization, in the area of integration processes, the throughput maximization is much more important than the optimization of the execution time of single process plan instances.

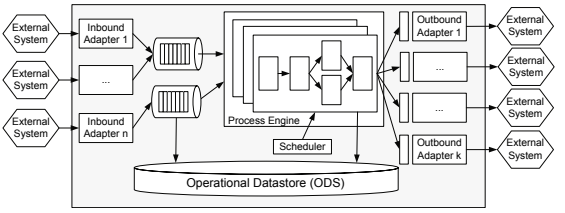
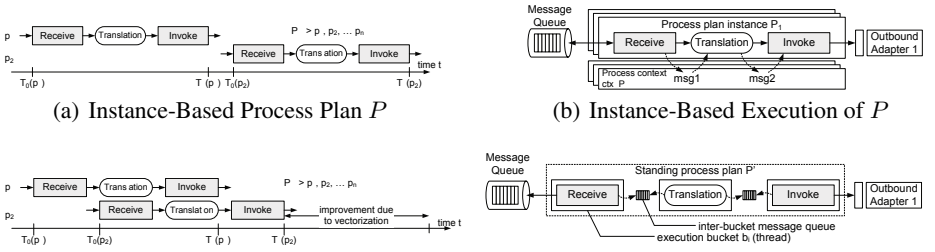


Fig. 1. Generalized Integration Platform Architecture

Due to the requirement of logical serialization of messages, those process plan instances cannot be executed in a multi-threaded way. As presented in the `SIR` transaction model [3], we must make sure that messages do not outrun other messages; for this purpose, we use logical serialization concepts such as message queues. The requirement of serialized execution of process plan instances is not always necessary. We can weaken this requirement to serialized external behavior of process plan instances, which allows us to apply a more fine-grained serialization concept. Finally, also the transactional behavior must be ensured using compensation- or recovery-based transaction models.

Based on the mentioned assumptions and requirements, we now formally define the integration process vectorization problem. Figure 2(a) illustrates the temporal aspects of a typical instance-based integration process. Semantically, in this example, a message is received from the inbound message queue (`Receive`), then a schema mapping (`Translation`) is processed and finally, the message is sent to an external system (`Invoke`). In this case, different instances of this process plan are serialized in incoming order. Such an instance-based process plan is the input of our vectorization problem. In contrast to this, Figure 2(c) shows the temporal aspects of a vectorized integration process. Here, only the external behavior (according to the start time T_0 and the end time T_1 of instances) must be serialized. Such a vectorized process plan is the output of the vectorization problem. This general problem is defined as follows.

Final edited form was published in "Advances in Databases and Information Systems: 13th East European Conference. Riga 2009", S. 253-269, ISBN 978-3-642-03973-7
https://doi.org/10.1007/978-3-642-03973-7_19



(a) Instance-Based Process Plan P (b) Instance-Based Execution of P

(c) Fully Vectorized Process Plan P' (d) Fully Vectorized Execution of P'

Fig. 2. Overview of Vectorization of Integration Processes

Definition 1. Integration Process Vectorization Problem (IPVP): *Let P denote a process plan and p_i with $p_i = (p_1, p_2, \ldots, p_n)$ denotes the implied process plan instances with $P \Rightarrow p_i$. Further, let each process plan P comprise a graph of operators $o_i = (o_1, o_2, \ldots, o_m)$. For serialization purposes, the process plan instances are executed in sequence with $T_1(p_i) \leq T_0(p_{i+1})$. Then, the integration process vectorization problem describes the search for the derived process plan P' that exhibits the highest degree of parallelism for the process plan instances p'_i such that the constraint conditions $(T_1(p'_i, o_i) \leq T_0(p'_i, o_{i+1})) \wedge (T_1(p'_i, o_i) \leq T_0(p'_{i+1}, o_i))$ hold and the semantic correctness is ensured.*

Based on the IPVP, we now recall the static cost analysis, where in general, cost denotes the execution time. If we assume an operator sequence o with constant operator costs $C(o_i) = 1$, clearly, the following costs exist

$$\begin{aligned} C(P) &= n \cdot m && \text{// instance-based} \\ C(P') &= n + m - 1 && \text{// fully vectorized} \\ \Delta(C(P) - C(P')) &= (n-1) \cdot (m-1) \end{aligned}$$

where n denotes the number of process plan instances and m denotes the number of operators. Clearly, this is an idealized model only used for illustration purposes. In practice, the improvement depends on the most time-consuming operator o'_k with $C(o'_k) = \max_{i=1}^{m} C(o'_i)$ of a vectorized process plan P', where the costs can be specified as follows:

$$\begin{aligned} C(P) &= n \cdot \sum_{i=1}^{m} C(o_i) \\ C(P') &= (n + m - 1) \cdot C(o'_k) \\ \Delta(C(P) - C(P')) &= n \cdot \sum_{i=1 \wedge i \neq k}^{m} C(o_i) - (n + m - 1) \cdot C(o'_k)\,. \end{aligned}$$

Obviously, the performance improvement can even be negative in case of a very small number of process plan instances n. However, over time—and hence, with an increasing n—the performance improvement grows linearly.

Final edited form was published in "Advances in Databases and Information Systems: 13th East European Conference. Riga 2009", S. 253-269, ISBN 978-3-642-03973-7
https://doi.org/10.1007/978-3-642-03973-7_19

The general idea is to rewrite the instance-based process plan—where each instance is executed as a thread—to a vectorized process plan, where each operator is executed as a single *execution bucket* and hence, as a single thread. Thus, we model a standing process plan. Due to different execution times of the single operators, inter-bucket queues (with max constraints[1]) are required for each data flow edge. Figures 2(b) and 2(d) illustrate those two different execution models. As already shown, this offers high optimization potential but this exclusively addresses the process engine, while all other components can be reused without changes. However, at the same time, major challenges have to be solved when transforming P into P'.

2.2 Message Model and Process Model

As a precondition for vectorization, our formal foundation—the *Message Transformation Model (MTM)* [4]—was extended in order to make it applicable also in the context of vectorized integration processes (then we refer to it as *VMTM*). Basically, the *MTM* consists of a message model and an instance-based process model.

We model a message m of a message type M as a quadruple with $m = (M, S, A, D)$, where M denotes the message type, S denotes the runtime state, and A denotes a map of atomic name-value attribute pairs with $a_i = (n, v)$. Further, D denotes a map of message parts, where a single message part is defined with $d_i = (n, t)$. Here, n denotes the part name and t denotes a tree of named data elements. In the VMTM, we extend it to a quintuple with $m = (M, C, S, A, D)$, where the context information C denotes an additional map of atomic name-value attribute pairs with $c_i = (n, v)$. This extension is necessary due to parallel message execution within one process plan.

A process plan P is defined with $P = (o, c, s)$ as a 3-tuple representation of a directed graph. Let o with $o = (o_1, \ldots, o_m)$ denote a sequence of operators, let c denote the context of P as a set of message variables msg_i, and let s denote a set of services $s = (s_1, \ldots, s_l)$. Then, an instance p_i of a process plan P, with $P \Rightarrow p_i$, executes the sequence of operators once. Each operator o_i has a specific type as well as an identifier NID (unique within the process plan) and is either of an *atomic* or of a *complex* type. Complex operators recursively contain sequences of operators with $o_i = (o_{i,1}, \ldots, o_{i,m})$. Further, an operator can have multiple input variables $msg_i \in c$, but only one output variable $msg_j \in c$. Each service s_i contains a type, a configuration and a set of operations. Further, we define a set of interaction-oriented operators *iop* (`Invoke`, `Receive` and `Reply`), control-flow-oriented operators *cop* (`Switch`, `Fork`, `Iteration`, `Delay` and `Signal`) and data-flow-oriented operators *dop* (`Assign`, `Translation`, `Selection`, `Projection`, `Join`, `Setoperation`, `Split`, `Orderby`, `Groupby`, `Window`, `Validate`, `Savepoint` and `Action`). Furthermore, in the *VMTM*, the flow relations between operators o_i do not specify the control flow but the explicit data flow in the form of message streams. Additionally, the `Fork` operator is removed because in the vectorized case, operators are model-inherently executed in parallel. Finally, we introduce the additional operators `AND` and `XOR` (for synchronization) as well as the `COPY` operator (for data flow splits).

[1] Due to different execution times of single operators, queues in front of cost-intensive operators include larger numbers of messages. In order to overcome the problem of high memory requirements, we constrained the maximal number of messages per queue.

Final edited form was published in "Advances in Databases and Information Systems: 13th East European Conference. Riga 2009", S. 253-269, ISBN 978-3-642-03973-7
https://doi.org/10.1007/978-3-642-03973-7_19

2.3 Rewriting Algorithm

Basically, we distinguish between unary (one input) and binary (multiple input) operators. Both unary and binary operators of an instance-based process plan can be rewritten with the same core concept (see [2] for the Algorithm) that contains the following four steps. First, we create a queue instance for each data dependency between two operators (the output message of operator o_i is the input message of operator o_{i+1}). Second, we create an execution bucket for each operator. Third, we connect each operator with the referenced input queue. Clearly, each queue is referenced by exactly one operator, but each operator can reference multiple queues. Fourth, we connect each operator with the referenced output queues. If one operator must be connected to n output queues with $n \geq 2$ (its results are used by multiple following operators), we insert a `Copy` operator after this operator. This `Copy` operator simply gets a message from one input queue, then copies it $n-1$ times and puts those messages into the n output queues. Although this rewriting algorithm is only executed once for all process instances, it is important to notice the cubic complexity with $O(m^3) = O(m^3+m^2)$, according to the number of operators m. This complexity is dominated by dependency checking when connecting operators and queues. Based on the standard rewriting concept, specific rewriting rules for context-specific operators (e.g., `Switch`) and for serialization and recoverability are required. Those rules and the related cost analysis are given in [2].

3 Cost-Based Vectorization

During rule-based process plan vectorization, an instance-based process plan (one execution bucket for all operators) is completely vectorized (one execution bucket for each operator). This solves the *integration process vectorization problem*. However, the two major weaknesses of this approach are (1) that the theoretical performance of a vectorized integration process mainly depends on the performance of the most cost-intensive operator and (2) that the practical performance also strongly depends on the number of available threads (and hence, on the number of operators). Thus, the optimality of process plan vectorization strongly depends on dynamic workload characteristics. For instance, the full process plan vectorization can also hurt performance due to additional thread management if the instance-based process plan has already caused a 100-percent resource consumption. In conclusion, we extend our approach and introduce a more generalized problem description and an approach for the cost-based vectorization of process plans. Obviously, the instance-based process plan and the fully vectorized process plan are specific cases of this more general solution.

3.1 Problem Generalization

The input (instance-based process plan) and the output (vectorized process plan) of the IPVP are extreme cases. In order to introduce awareness of dynamically changing workload characteristics, we generalize the IPVP to the Cost-Based IPVP as follows:

Definition 2. Cost-Based Integration Process Vectorization Problem (CBIPVP): *Let P denote a process plan and p_i with $p_i = (p_1, p_2, \ldots, p_n)$ denotes the implied process plan instances with $P \Rightarrow p_i$. Further, let each process plan P comprise a graph*

Final edited form was published in "Advances in Databases and Information Systems: 13th East European Conference. Riga 2009", S. 253-269, ISBN 978-3-642-03973-7
https://doi.org/10.1007/978-3-642-03973-7_19

Table 1. Example Operator Distribution

		b_1	b_2	b_3	b_4
1	$k=1$	o_1, o_2, o_3, o_4	-	-	-
2	$k=2$	o_1	o_2, o_3, o_4	-	-
3		o_1, o_2	o_3, o_4	-	-
4		o_1, o_2, o_3	o_4	-	-
5	$k=3$	o_1	o_2	o_3, o_4	-
6		o_1	o_2, o_3	o_4	-
7		o_1, o_2	o_3	o_4	-
8	$k=4$	o_1	o_2	o_3	o_4

of operators $o_i = (o_1, o_2, \ldots, o_m)$. For serialization purposes, the process plan instances are executed in sequence with $T_1(p_i) \leq T_0(p_{i+1})$. The CBIPVP describes the search for the derived cost-optimal (minimal execution time of a message sequence) process plan P'' with $k \in \mathbb{N}$ execution buckets $b_i = (b_1, b_2, \ldots, b_k)$, where each bucket contains l operators $o_i = (o_1, o_2, \ldots, o_l)$. Here, the constraint conditions $(T_1(p''_i, b_i) \leq T_0(p''_i, b_{i+1})) \wedge (T_1(p''_i, b_i) \leq T_0(p''_{i+1}, b_i))$ and $(T_1(b_i, o_i) \leq T_0(b_i, o_{i+1})) \wedge (T_1(b_i, o_i) \leq T_0(p''_{i+1}, b_i))$ must hold. We define that $(l_{b_i} \geq 1) \wedge (l_{b_i} \leq m)$ and $\sum_{i=1}^{|b_i|} l_{b_i} = m$ and that each operator o_i is assigned to exactly one bucket b_i.

If we reconsider the IPVP, on the one hand, an instance-based process plan P is a specific case of the cost-based vectorized process plan P'', with $k = 1$ execution bucket. On the other hand, also the fully vectorized process plan P' is a specific case of the cost-based vectorized process plan P'', with $k = m$ execution buckets, where m denotes the number of operators o_i. The following example illustrates that problem.

Example 1. Operator distribution across buckets: *Assume a simple process plan P with a sequence of four operators ($m = 4$). Table 1 shows the possible process plans for the different numbers of buckets k. We can distinguish eight different ($2^{4-1} = 8$) process plans. Here, plan 1 is the special case of an instance-based process plan and plan 8 is the special case of a fully vectorized process plan.*

Theorem 1. *The cost-based integration process vectorization problem exhibits an exponential complexity of $O(2^m)$.*

Proof. The distribution function $\mathcal{D}$ of the number of possible plans over k is a symmetric distribution function according to *Pascal's Triangle*, where the condition $l_{b_i} = l_{b_{k-i+1}}$ with $i \leq \frac{m}{2}$ does hold. Based on Definition 1, a process plan contains m operators. Due to Definition 2, we search for k execution buckets b_i with $l_{b_i} \geq 1 \wedge l_{b_i} \leq m$ and $\sum_{i=1}^{|b_i|} l_{b_i} = m$. Hence, m ($k = 1, ..., k = m$) different numbers of buckets have to be evaluated. From now on, we fix m' as $m' = m - 1$ and k' as $k' = k - 1$. In fact, there is only one possible plan for $k = 1$ (all operators in one bucket) and $k = m$ (each operator in a different bucket), respectively.

$$|P|_{k'=0} = \binom{m'}{0} = 1 \quad \text{and} \quad |P|_{k'=m'} = \binom{m'}{m'} = 1 \,.$$

Now, fix a specific m and k. Then, the number of possible plans is computed with

Final edited form was published in "Advances in Databases and Information Systems: 13th East European Conference. Riga 2009", S. 253-269, ISBN 978-3-642-03973-7
https://doi.org/10.1007/978-3-642-03973-7_19

$$|P|_k = \binom{m'}{k'} = \binom{m'-1}{k'-1} + \binom{m'-1}{k'} = \prod_{i=1}^{k'} \frac{m'+1-i}{i}.$$

In order to compute the number of possible plans, we have to sum up the possible plans for each k, with $1 \leq k \leq m$:

$$|P| = \sum_{k'=0}^{m'} \binom{m'}{k'} \text{ with } k' = k-1 \text{ and } m' = m-1 \,.$$

Finally, $\sum_{k=0}^{n} \binom{n}{k}$ is known to be equal to 2^n. Hence, by changing the index k from $k' = 0$ to $k = 1$ we can write:

$$|P| = \sum_{k'=0}^{m'} \binom{m'}{k'} = \sum_{k=1}^{m} \binom{m-1}{k-1} = 2^{(m-1)} \,.$$

In conclusion, there are $2^{(m-1)}$ possible process plans that must be evaluated. Due to the linear complexity of $O(m)$ for determining the costs of a plan, the cost-based integration process vectorization problem exhibits an exponential overall complexity with $O(2^m) = O(m \cdot 2^{(m-1)})$.

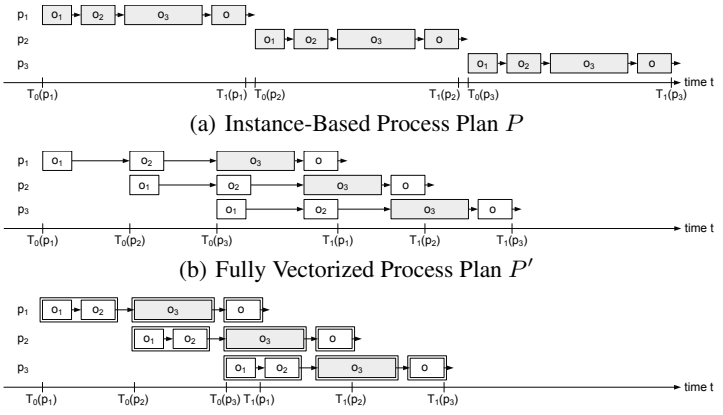
Note that we have a recursive algorithm because we need to include complex operators as well. For understandability, we simplified this to a sequence of atomic operators.

3.2 Heuristic Approach

Due to the exponential complexity of the CBIPVP, a search space reduction approach for determining the cost-optimal solution for the CBIPVP is strongly needed. Here, we present a heuristic-based algorithm that solves the CBIPVP with linear complexity of $O(m)$. The core heuristic of our approach is illustrated in Figure 3. Basically, we set $k = m$, where each operator is executed in a single execution bucket. Then, we merge those execution buckets in a cost-based fashion. Typically, the improvements achieved by vectorization mainly depend on the most time-consuming operator o_k with $C(P, o_k) = \max_{i=1}^{m} C(P, o_i)$ of a process plan P. The reason is that the costs of a vectorized process plan are computed with $C(P') = (n+m-1) \cdot C(o'_k)$, and hence, the work cycle of the vectorized process plan is given by $C(o'_k)$. Thus, the time period of the start of two subsequent process plan instances is given by $T_0(p_{i+1}) - T_0(p_i) = C(o'_k)$. It would be more efficient to leverage the queue waiting times and merge execution buckets. Hence, we use this heuristic to solve the constrained problem.

Definition 3. Constrained CBIPVP: *According to the CBIPVP, find the minimal number of buckets k and an assignment of operators o_i with $i = 1, .., m$ to those execution buckets b_i with $i = 1, ..k$ such that $\forall b_i : C(b''_i) \leq \max_{i=1}^{m} C(o'_i) + \lambda$, and make sure that the assignment preserves the order with respect to the operator sequence o. Here, λ is a user-defined parameter (in terms of execution time) to control the cost constraint.*

Final edited form was published in "Advances in Databases and Information Systems: 13th East European Conference. Riga 2009", S. 253-269, ISBN 978-3-642-03973-7
https://doi.org/10.1007/978-3-642-03973-7_19



(a) Instance-Based Process Plan P

(b) Fully Vectorized Process Plan P'

(c) Cost-Based Vectorized Process Plan P''

Fig. 3. Work Cycle Domination by Operator o_3

Figure 4 illustrates the influence of λ and the core idea of solving the constrained problem. Each operator o_i has assigned costs $C(o'_i)$. In our example, it is $\max_{i=1}^{m} C(o'_i) = C(o'_3) = 5$. The Constrained CBIPVP describes the search for the minimal number of execution buckets, where the cumulative costs of each bucket must not be larger than the determined maximum plus a user-defined λ. Hence, in our example, we search for the k buckets, where the cumulative costs of each bucket are, at most,

Algorithm 1. Cost-Based Bucket Determination

Require: operator sequence o
1: $A \leftarrow \oslash, B \leftarrow \oslash, k \leftarrow 0$
2: $max \leftarrow \max_{i=1}^{m} C(P', o_i) + \lambda$
3: **for** $i = 1$ to $|o|$ **do** *// foreach operator*
4: **if** $o_i \in A$ **then**
5: **continue** 3
6: **end if**
7: $k \leftarrow k + 1$
8: $b_k(o_i) \leftarrow$ create bucket over o_i
9: **for** $j = i + 1$ to $|o|$ **do** *// foreach following operator*
10: **if** $\sum_{c=1}^{|b_k|} C(o_c) + C(o_j) \leq max$ **then**
11: $b_k \leftarrow$ add o_j to b_k
12: $A \leftarrow A \cup o_j$
13: **else**
14: **break** 9
15: **end if**
16: **end for**
17: $B \leftarrow B \cup b_k$
18: **end for**
19: **return** B

Final edited form was published in "Advances in Databases and Information Systems: 13th East European Conference. Riga 2009", S. 253-269, ISBN 978-3-642-03973-7
https://doi.org/10.1007/978-3-642-03973-7_19

equal to five. If we increase λ, we can reduce the number of buckets by increasing the allowed maximum and hence, the work cycle of the vectorized process plan.

Algorithm 2 illustrates the concept of the cost-based bucket determination algorithm. Here, the operator sequence[2] o is required. First, we initialize two sets A and B as empty sets. After that, we compute the maximal costs of a bucket max with $max = \max_{i=1}^{m} C(o_i') + \lambda$. Then, there is the main loop over all operators. If the operator o_i belongs to A (operators already assigned to buckets), we can proceed with the next operator. Otherwise, we create a new bucket b_k and increment k (number of buckets) accordingly. After that, we execute the inner loop in order to assign operators to this bucket such that the constraint $\sum_{c=1}^{|b_k|} C(o_c) \leq max$ holds. This is done by adding o_j to b_k and to A. Here, we can ensure that each created bucket has at least one operator assigned. Finally, each new bucket b_k is added to the set of buckets B.

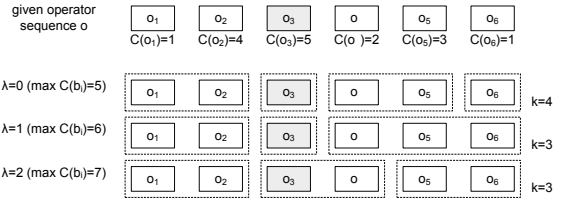


Fig. 4. Bucket Merging with Different λ

Theorem 2. *The cost-based bucket determination algorithm solves the constrained cost-based integration process vectorization problem with linear complexity of* $O(m)$.

Proof. Assume a process plan that comprises a sequence of m operators. First, the maximum of a value list (line 2) is known to be of complexity $O(m)$ (m operator evaluations). Second, we can see that the bucket number is at least 1 (all operators assigned to one bucket) and at most m (each operator assigned to exactly one bucket). Third, in the case of $k = 1$ there are at most $2m - 1$ possible operator evaluations. Also, in the case of $k = m$ there are at most $2m - 1$ possible operator evaluations. If we assume that the operations $\in$ and $\cup$—in our case—exhibit constant time complexity of $O(1)$, we now can conclude that the cost-based bucket determination algorithm exhibits a linear complexity with $O(m) = O(3m - 1) = O(m) + O(2m - 1)$.

3.3 Optimality Analysis

As already mentioned, the optimality of the vectorized process plan depends on (1) the costs of the single operators, (2) the resource consumption of each operator and (3) the available hardware resources (possible parallelism). However, the cost-based bucket determination algorithm only takes the costs from (1) into consideration. Nevertheless, we show that optimality guarantees can be given using this heuristic approach.

The algorithm can be parameterized with respect to the hardware resources (3). If we want to force a single-threaded execution, we simply set λ to $\lambda \geq \sum_{i=1}^{m} C(o_i) - \max_{i=1}^{m} C(o_i)$. If we want to force the highest meaningful degree of parallelism (this is not necessarily a full vectorization), we simply set $\lambda = 0$.

[2] Note that each process plan is a sequence of atomic and complex operators. Due to those complex operators, the cost-based bucket determination algorithm is a recursive algorithm. However, we transformed it to a linear one in order to show the core concept.

Final edited form was published in "Advances in Databases and Information Systems: 13th East European Conference. Riga 2009", S. 253-269, ISBN 978-3-642-03973-7
https://doi.org/10.1007/978-3-642-03973-7_19

Now, assuming the given λ configuration, the question is, which optimality guarantee can we give for the solution of the cost-based bucket determination algorithm. For this purpose, $R_e(o_i)$ denotes the empirical resource consumption (measured with a specific configuration) of an operator o_i with $0 \leq R_e(o_i) \leq 1$, and $R_o(o_i)$ denotes the maximal resource consumption of an operator o_i with $0 \leq R_o(o_i) \leq 1$. Here, $R_o(o_i) = 1$ means that the operator o_i exhibits an average resource consumption of 100 percent. In fact, the condition $\sum_{i=1}^{m} R_e(o_i) \leq 1$ must hold.

Obviously, for an instance-based plan P, we can write $R_e(o_i) = R_o(o_i)$ because all operators are executed in sequence. When we vectorize P to a fully vectorized plan P', with a maximum of $R_e(o'_i) = \frac{1}{m}$, we have to compute the costs with $C(o'_i) = \frac{R_o(o_i)}{R_e(o'_i)} \cdot C(o_i)$. When we merge two execution buckets b'_i and b'_{i+1} during cost-based bucket determination, we compute the effective resource consumption $R_e(b''_i) = \frac{1}{|b|-1}$, the maximal resource consumption $R_o(b''_i) = \frac{C(b'_i) \cdot R_o(b'_i) + C(b'_{i+1}) \cdot R_o(b'_{i+1})}{C(b'_i) + C(b'_{i+1})}$, and the cost

$$C(b''_i) = \begin{cases} \frac{R_e(b'_i)}{R_e(b''_i)} \cdot C(b'_i) + \frac{R_e(b'_{i+1})}{R_e(b''_{i+1})} \cdot C(b'_{i+1}) & R_e(b''_i) \leq R_o(b''_i) \\ \frac{R_o(b'_i)}{R_o(b''_i)} \cdot C(b'_i) + \frac{R_o(b'_{i+1})}{R_o(b''_{i+1})} \cdot C(b'_{i+1}) & otherwise \end{cases}.$$

Obviously, we made the assumption that each execution bucket gets the same maximal resource consumption $R_e(b''_i)$ and that resources are not exchanged between those buckets. We do not take the temporal overlap into consideration. However, we can give the following optimality guarantee.

Theorem 3. *The cost-based bucket determination algorithm solves the cost-based integration process vectorization problem with an optimality guarantee of* $(C(P'') \leq C(P)) \wedge (C(P'') \leq C(P'))$ *under the restriction of* $\lambda = 0$.

Proof. As a precondition it is important to notice that the cost-based bucket determination algorithm cannot result in a plan with $k = 1$ (although this is a special case of the CBIPVP) due to the maximum rule of $\sum_{c=1}^{|b_k|} C(o_c) + C(o_j) \leq max$ (Algorithm 2, line 10). Hence, in order to prove the theorem, we only need to prove the two single claims of $C(P'') \leq C(P)$ and $C(P'') \leq C(P')$.

For the proof of $C(P'') \leq C(P)$, assume the worst case where $\forall o_i \in R_o(o_i) = 1$. If we vectorize this to P'', we need to compute the costs by $C(b''_i) = \frac{R_o(b''_i)}{R_e(b''_i)} \cdot C(o_i)$ with $R_e(b''_i) = \frac{1}{m}$. Due to the vectorized execution, $C(P'') = \max_{i=1}^{m} C(b''_i)$, while $C(P) = \sum_{i=1}^{m} C(o_i)$. Hence, we can write $C(P'') = C(P)$ if the condition $\forall o_i \in R_o(o_i) = 1$ holds. This is the worst case. For each $R_o(o_i) < 1$, we get $C(P'') < C(P)$.

In order to prove $C(P'') \leq C(P')$, we set $\lambda = 0$. If we merge two buckets b_i and b_{i+1}, we see that $R_e(b''_i)$ is increased from $\frac{1}{|b|}$ to $\frac{1}{|b|-1}$. Thus, we re-compute the costs $C(b''_i)$ as mentioned before. In the worst case, $C(b''_i) = C(b'_i)$, which is true iff $R_e(b'_i) = R_o(b'_i)$ because then we also have $R_e(b''_i) = R_e(b'_i)$. Due to $C(P'') = \max_{i=1}^{m} C(b''_i)$, we can state $C(P'') \leq C(P)$. Hence, the theorem holds.

In conclusion, we cannot guarantee that the result of the cost-based bucket determination algorithm is the global optimum because we cannot evaluate the effective resource

Final edited form was published in "Advances in Databases and Information Systems: 13th East European Conference. Riga 2009", S. 253-269, ISBN 978-3-642-03973-7
https://doi.org/10.1007/978-3-642-03973-7_19

consumption in an efficient way. However, we can guarantee that each merging of execution buckets when solving the CBIPVP improves the performance of the process plan P. Hence, we follow a best-effort optimization approach.

3.4 Dynamic Process Plan Rewriting

Due to the cost-based bucket determination approach, the dynamic process plan rewriting is required. The major problem when rewriting a vectorized process plan during runtime is posed by loaded queues. The used queues can be stopped using the `stopped` flag. If we—for example—want to merge two execution buckets b_i and b_{i+1}, we need to stop the queue q_i that is represented by the edge between b_{i-1} and b_i. Then, we wait until the queue q_{i+1} (the queue just between b_i and b_{i+1}) contains 0 messages. Now, we can merge the execution buckets to b_i and simply remove q_{i+1}. This concept can be used for bucket merging and splitting, respectively. Finally, the rewriting algorithm is triggered only if a different plan than the current one has been determined by the cost-based bucket determination algorithm. In fact, we need to compare two plans P_1'' and P_2'' by graph matching. However, this is known to be of linear complexity with $O(m)$.

4 Experimental Evaluation

In this section, we provide selected experimental results. In general, we can state that the vectorization of integration processes leads to a significant performance improvement. In fact, we can show that Theorem 4 ($C(P'') \leq C(P) \wedge C(P'') \leq C(P')$ under the restriction of $\lambda = 0$) does also hold during experimental performance evaluation.

4.1 Experimental Setup

We implemented the presented approach within our so-called `WFPE` (workflow process engine) using Java 1.6 as the programming language. Here, we give a brief overview of the `WFPE` and discuss some implementation details. In general, the `WFPE` uses compiled process plans (a java class is generated for each integration process type). Furthermore, it follows an instance-based execution model. Now, we integrated components for the static vectorization of integration processes (we call this `VWFPE`) and for the cost-based vectorization (we call this `CBVWFPE`). For that, new deployment functionalities were introduced (those processes are executed in an interpreted fashion) and several changes in the runtime environment were required. Finally, all three different runtime approaches can be used alternatively.

We ran our experiments on a standard blade (OS Suse Linux) with two processors (each of them a Dual Core AMD Opteron Processor 270 at 1,994 MHz) and 8.9 GB RAM. Further, we executed all experiments on synthetically generated XML data (using our DIPBench toolsuite [5]) because the data distribution of real data sets has only minor influence on the performance of the integration processes used here. However, there are several aspects that influence the performance improvement of the vectorization and hence, these should be analyzed. In general, we used the following five aspects as scale factors for all three execution approaches: data size d of a message, the number

Final edited form was published in "Advances in Databases and Information Systems: 13th East European Conference. Riga 2009", S. 253-269, ISBN 978-3-642-03973-7
https://doi.org/10.1007/978-3-642-03973-7_19

of nodes m of a process plan, the time interval t between two messages, the number of process instances n and the maximal number of messages q in a queue. Here, we measured the performance of different combinations of those. For statistical correctness, we repeated all experiments 20 times and computed the arithmetic mean.

As base integration process for our experiments, we modeled a simple sequence of six operators. Here, a message is received (`Receive`) and then an archive writing is prepared (`Assign`) and executed with the file adapter (`Invoke`). After that, the resulting message (contains Orders and Orderlines) is translated using an XML transformation (`Translation`) and finally sent to a specific directory (`Assign`, `Invoke`). We refer to this as $m = 5$ because the `Receive` is removed during vectorization. When scaling m up to $m = 35$, we simply copy the last five operators and reconfigure them.

4.2 Performance and Throughput

Here, we ran a series of five experiments according to the already introduced influencing aspects. The results of these experiments are shown in Figure 5. Basically, the five experiments correlate to the mentioned scale factors.

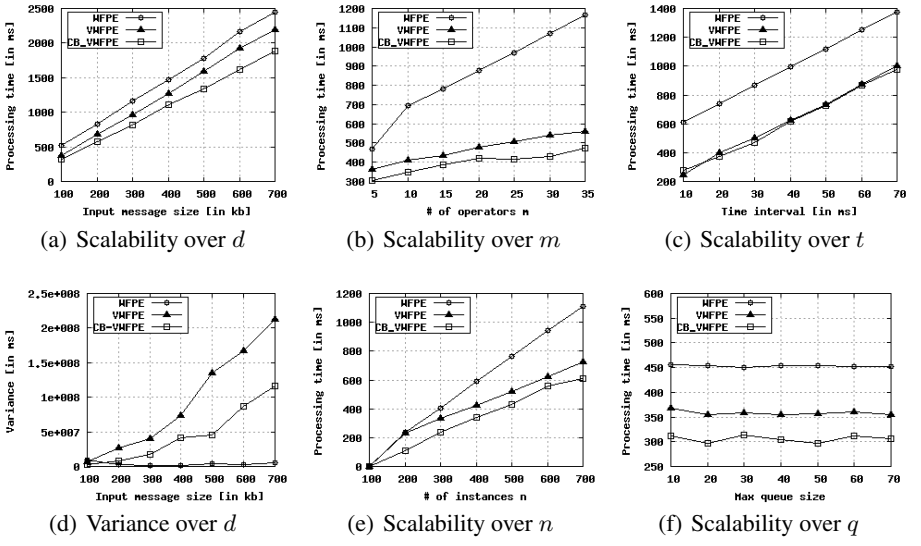
In Figure 5(a), we scaled the data size d of the XML input messages from 100 kb to 700 kb and measured the processing time for 250 process instances ($n = 250$) needed by the three different runtimes. There, we fix $m = 5$, $t = 0$, $n = 250$ and $q = 50$. We can observe that all three runtimes exhibit a linear scaling according to the data size and that significant improvements can be reached using vectorization. There, the absolute improvement increases with increasing data size. Further, in Figure 5(d), we illustrated the variance of this sub-experiment. The variance of the instance-based execution is minimal, while the variances of both vectorized runtimes are worse because of the operator scheduling. Note that the cost-based vectorization exhibits a significantly lower variance than in the fully vectorized case because of a lower number of threads.

Now, we fix $d = 100$ (lowest improvement in 5(a)), $t = 0$, $n = 250$ and $q = 50$ in order to investigate the influence of m. We vary m from 5 to 35 nodes as already mentioned for the experimental setup. Interestingly, not only the absolute but also the relative improvement of vectorization increases with increasing number of operators. In comparison to full vectorization, for cost-based vectorization, a constant relative improvement is observable.

Figure 5(c) shows the impact of the time interval t between the initiation of two process instances. For that, we fix $d = 100$, $m = 5$, $n = 250$, $q = 50$ and vary t from 10 ms to 70 ms. There is almost no difference between the full vectorization and the cost-based vectorization. However, the absolute improvement between instance-based and vectorized approaches decreases slightly with increasing t. An explanation is that the time interval has no impact on the instance-based execution. In contrast to that, the vectorized approach depends on t because of resource scheduling.

Further, we analyze the influence of the number of instances n as illustrated in Figure 5(e). Here, we fix $d = 100$, $m = 5$, $t = 0$, $q = 50$ and vary n from 100 to 700. Basically, we can observe that the relative improvement between instance-based and vectorized execution increases when increasing n, due to parallelism of process instances. However, it is interesting to note that the fully vectorized solution performs

Final edited form was published in "Advances in Databases and Information Systems: 13th East European Conference. Riga 2009", S. 253-269, ISBN 978-3-642-03973-7
https://doi.org/10.1007/978-3-642-03973-7_19



(a) Scalability over d (b) Scalability over m (c) Scalability over t

(d) Variance over d (e) Scalability over n (f) Scalability over q

Fig. 5. Experimental Performance Evaluation Results

slightly better for small n. However, when increasing n, the cost-based vectorized approach performs optimal.

Figure 5(f) illustrates the influence of the maximal queue size q, which we varied from 10 to 70. Here, we fix $d = 100$, $m = 5$, $t = 0$ and $n = 250$. In fact, q slightly affects the overall performance for a small number of concurrent instances n. However, at $n = 250$, we cannot observe any significant influence with regard to the performance.

4.3 Deployment and Maintenance

The purpose of this experiment was to analyze the deployment overhead of three different runtimes. We measured the costs for the process plan vectorization (PPV) algorithm and the periodically invoked cost-based bucket determination (CBBD) algorithm.

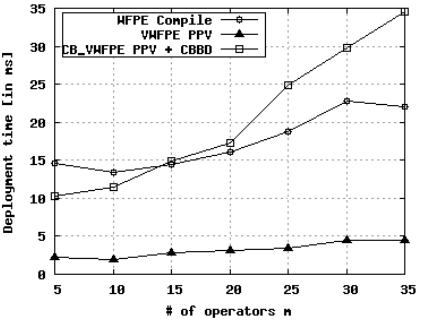


Fig. 6. Vectorization Overhead Analysis

Figure 6 shows those results. Here, we varied the number of nodes m because all other scale factors do not influence the deployment and maintenance costs. In general, there is a huge performance improvement using vectorization with a factor of up to seven. It is caused by the different deployment approaches. The `WFPE` uses a compilation approach, where java classes are generated from the integration process specification. In contrast to this, the

Final edited form was published in "Advances in Databases and Information Systems: 13th East European Conference. Riga 2009", S. 253-269, ISBN 978-3-642-03973-7
https://doi.org/10.1007/978-3-642-03973-7_19

VWFPE as well as the CBVWFPE use interpretation approaches, where process plans are built dynamically with the PPV algorithm. In fact, VWFPE always outperforms CBVWFPE because both use the PPV algorithm but CBVWFPE additionally uses the CBBD algorithm in order to find the optimal k. Note that the additional costs for the CBBD algorithm (that cause a break-even point with the standard WFPE) occur periodically during runtime (period is a parameter). In conclusion, the vectorization of integration processes allows for better runtime as well as better deployment time performance. Hence, this approach can be used under all circumstances.

5 Related Work

We review related work from the perspectives of computational engineering, database management systems, data stream management systems, streaming service and process execution, and integration process optimization.

Computational Engineering. Based on Flynn's classification, vectorization can be classified as SIMD (single instruction, multiple data) and in special cases as MIMD (multiple instruction, single data). Here, we use the term vectorization only as analogy.

Database Management Systems. In the context of DBMS, throughput optimization has been addressed with different techniques. One significant approach is data sharing across common subexpressions of different queries or query instances [6,7,8,9]. However, in [10] it was shown that sharing can also hurt performance. Another inspiring approach is given by staged DBMS [11]. Here, in the QPipe Project [12,13], each relational operator was executed as a so-called micro-engine (one operator, many queries). Further, throughput optimization approaches were introduced also in the context of distributed query processing [14,15].

Data Stream Management Systems. Further, in the context of data stream management systems (DSMS) and ETL tools, the *pipes-and-filters* execution model is widely used. Examples for those systems are QStream [16], Demaq [17] and Borealis [18]. Surprisingly, the cost-based vectorization has not been used so far because the operator scheduling [19,20,21,22] in DSMS is not realized with multiple processes or threads but with central control strategies (assuming high costs for switching the process context). Furthermore, there is one interesting approach [23], where operators are distributed across a number of threads in a query-aware manner. However, this approach does not compute the cost-optimal distribution.

Streaming Service and Process Execution. In service-oriented environments, throughput optimization has been addressed on different levels. Performance and resource issues, when processing large volumes of XML documents, lead to message chunking on service-invocation level. There, request documents are divided into chunks and services are called for every single chunk [24]. An automatic chunk-size computation using the extremum-control approach was addressed in [25]. On process level, pipeline scheduling was incorporated in [26] into a general workflow model to show the valuable benefit of pipelining in business processes. Further, [1] adds pipeline semantics to classic step-by-step workflows by extending available task states and utilizing a one-item queue between two consecutive tasks. None of those approaches deals with cost-based rewriting of instance-based processes to pipeline semantics.

Final edited form was published in "Advances in Databases and Information Systems: 13th East European Conference. Riga 2009", S. 253-269, ISBN 978-3-642-03973-7
https://doi.org/10.1007/978-3-642-03973-7_19

Integration Process Optimization. Optimization of integration processes has not yet been explored sufficiently. There are platform-specific optimization approaches for the pipes-and-filters execution model, like the optimization of ETL processes [27], as well as numerous optimization approaches for instance-based processes like the optimization of data-intensive decision flows [28], the static optimization of the control flow, the use of critical path approaches [29] and SQL-supporting BPEL activities and their optimization [30]. Further, we investigated the optimization of message transformation processes [4] and the cost-based optimization of instance-based integration processes [31]. Finally, the rule-based vectorization approach—presented in [2]—was the foundation of our cost-optimal solution to the vectorization problem.

6 Conclusions

In order to optimize the throughput of integration platforms, in this paper, we revisited the concept of automatic vectorization of integration processes. Due to the dependence on the dynamic workload characteristics, we introduced the cost-based process plan vectorization, where the costs of single operators are taken into account and operators are merged to execution buckets. Based on our experimental evaluation, we can state that significant throughput improvements are possible. In conclusion, the concept of process vectorization is applicable in many different application areas. Future work can address specific optimization techniques for the cost-based vectorization.

References

1. Biornstad, B., Pautasso, C., Alonso, G.: Control the flow: How to safely compose streaming services into business processes. In: IEEE SCC (2006)
2. Boehm, M., Habich, D., Lehner, W., Wloka, U.: Vectorizing instance-based integration processes. In: ICEIS (2009), http://wwwdb.inf.tu-dresden.de/team/archives/2007/04/dipl_wirtinf_ma.php
3. Boehm, M., Habich, D., Lehner, W., Wloka, U.: An advanced transaction model for recovery processing of integration processes. In: ADBIS (2008)
4. Boehm, M., Habich, D., Wloka, U., Bittner, J., Lehner, W.: Towards self-optimization of message transformation processes. In: ADBIS (2007)
5. Boehm, M., Habich, D., Lehner, W., Wloka, U.: Dipbench toolsuite: A framework for benchmarking integration systems. In: ICDE (2008)
6. Dalvi, N.N., Sanghai, S.K., Roy, P., Sudarshan, S.: Pipelining in multi-query optimization. In: PODS (2001)
7. Hasan, W., Motwani, R.: Optimization algorithms for exploiting the parallelism-communication tradeoff in pipelined parallelism. In: VLDB (1994)
8. Roy, P., Seshadri, S., Sudarshan, S., Bhobe, S.: Efficient and extensible algorithms for multi query optimization. In: SIGMOD (2000)
9. Wilschut, A.N., van Gils, S.A.: A model for pipelined query execution. In: MASCOTS (1993)
10. Johnson, R., Hardavellas, N., Pandis, I., Mancheril, N., Harizopoulos, S., Sabirli, K., Ailamaki, A., Falsafi, B.: To share or not to share? In: VLDB (2007)
11. Harizopoulos, S., Ailamaki, A.: A case for staged database systems. In: CIDR (2003)

Final edited form was published in "Advances in Databases and Information Systems: 13th East European Conference. Riga 2009", S. 253-269, ISBN 978-3-642-03973-7
https://doi.org/10.1007/978-3-642-03973-7_19

12. Gao, K., Harizopoulos, S., Pandis, I., Shkapenyuk, V., Ailamaki, A.: Simultaneous pipelining in qpipe: Exploiting work sharing opportunities across queries. In: ICDE (2006)
13. Harizopoulos, S., Shkapenyuk, V., Ailamaki, A.: Qpipe: A simultaneously pipelined relational query engine. In: SIGMOD (2005)
14. Ives, Z.G., Florescu, D., Friedman, M., Levy, A.Y., Weld, D.S.: An adaptive query execution system for data integration. In: SIGMOD (1999)
15. Lee, R., Zhou, M., Liao, H.: Request window: an approach to improve throughput of rdbms-based data integration system. In: VLDB (2007)
16. Schmidt, S., Berthold, H., Lehner, W.: Qstream: Deterministic querying of data streams. In: VLDB (2004)
17. Boehm, A., Marth, E., Kanne, C.C.: The demaq system: declarative development of distributed applications. In: SIGMOD (2008)
18. Abadi, D.J., Ahmad, Y., Balazinska, M., Çetintemel, U., Cherniack, M., Hwang, J.H., Lindner, W., Maskey, A., Rasin, A., Ryvkina, E., Tatbul, N., Xing, Y., Zdonik, S.B.: The design of the borealis stream processing engine. In: CIDR (2005)
19. Babcock, B., Babu, S., Datar, M., Motwani, R., Thomas, D.: Operator scheduling in data stream systems. VLDB J. 13(4) (2004)
20. Carney, D., Çetintemel, U., Rasin, A., Zdonik, S.B., Cherniack, M., Stonebraker, M.: Operator scheduling in a data stream manager. In: VLDB (2003)
21. Koch, C., Scherzinger, S., Schweikardt, N., Stegmaier, B.: Schema-based scheduling of event processors and buffer minimization for queries on structured data streams. In: VLDB (2004)
22. Schmidt, S., Legler, T., Schaller, D., Lehner, W.: Real-time scheduling for data stream management systems. In: ECRTS (2005)
23. Cammert, M., Heinz, C., Krämer, J., Seeger, B., Vaupel, S., Wolske, U.: Flexible multi-threaded scheduling for continuous queries over data streams. In: ICDE Workshops (2007)
24. Srivastava, U., Munagala, K., Widom, J., Motwani, R.: Query optimization over web services. In: VLDB (2006)
25. Gounaris, A., Yfoulis, C., Sakellariou, R., Dikaiakos, M.D.: Robust runtime optimization of data transfer in queries over web services. In: ICDE (2008)
26. Lemos, M., Casanova, M.A., Furtado, A.L.: Process pipeline scheduling. J. Syst. Softw. 81(3) (2008)
27. Simitsis, A., Vassiliadis, P., Sellis, T.: Optimizing etl processes in data warehouses. In: ICDE (2005)
28. Hull, R., Llirbat, F., Kumar, B., Zhou, G., Dong, G., Su, J.: Optimization techniques for data-intensive decision flows. In: ICDE (2000)
29. Li, H., Zhan, D.: Workflow timed critical path optimization. Nature and Science 3(2) (2005)
30. 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)
31. Boehm, M., Habich, D., Lehner, W., Wloka, U.: Workload-based optimization of integration processes. In: CIKM (2008)