Efficient and Distributed Rule Placement in Heavy Constraint-Driven Event Systems

Björn Schilling, Boris Koldehofe, and Kurt Rothermel
Institute of Parallel and Distributed Systems
Universitätsstr. 38, Stuttgart, Germany
$schilling, koldehofe, rothermel$@ipvs.uni-stuttgart.de

***Abstract*—Complex Event Processing (CEP) is of increasing importance in many industrial applications to integrate a huge number of events in a scalable manner. A core challenge towards scalable CEP is to efficiently distribute the rules which define how correlations between events can be detected within an event processing network. Although significant progress has been made recently, there remains a fundamental gap in supporting requirements that emerge from deploying CEP over heterogeneous and independent processing environments. Heterogeneity typically imposes many constraints on the placement of rules, which increases the complexity of the underlying optimization problem and cannot be handled efficiently by existing solutions.**

In this paper we examine the distributed placement, migration and optimization of rules in the context of the constraint optimization problem to minimize network usage. We propose and evaluate a placement algorithm that efficiently finds valid solutions in scenarios where the solution space is heavily restricted by constraints. The algorithm operates in a decentralized way and is adaptive to dynamic changes of processing nodes, rules, and load characteristics of the event processing network. The proposed rule migration policies resolve invalid placements quickly and thus ensure high availability. The evaluations show that the proposed algorithm is able to efficiently find near optimum solutions within heavy constraint-driven network conditions.

***Index Terms*—Heterogeneous Systems and Networks, Distributed Complex Event Processing, Optimization, Placement**

I. Introduction

Sensor devices, web services, business processes or other event sources are creating enormous and increasing numbers of event messages every day. Complex Event Processing (CEP) detects situations based on these events by correlating event messages and thus composes higher value information (cf. [5], [2], [11]). This correlation process is described by CEP users as a set of correlation rules. Many classical event processing solutions, which detect situations on a single engine, are reaching their limits as both the computation and communication costs are increasing alongside. As a result, a significant amount of research has been conducted in the field of distributed CEP. Here, the detection of a situation is performed by multiple nodes (and engines) within the network. The distribution of event correlation enables scalable applications and can also result in a more efficient CEP: for example by moving the functionality closer to the sources, the network usage of detecting situations can be improved. Furthermore, by distributing the same functionality among several nodes, CEP applications become more available and reliable.

The distribution of multiple rules within the *network of correlation engines* (further referred to as *CEP system*) raises the question of placement. Consider a large power grid, where several participants (e.g. energy producers, brokers, customers - further referred to as *domains*) are producing, trading or consuming energy. In this scenario, a set of rules is defined to detect energy load violations, regulate energy consumption or steering the energy flow. The challenge is to find a placement of the rules, such that CEP is efficient in respect to the applications optimization goals, such as network usage.

The placement problem in event processing systems has been tackled by many researchers recently (cf. [6], [11], [12]). However, so far, distributed CEP is characterized by relying on a homogeneous system, where all nodes have the same capabilities and no restrictions are imposed on the placement. Handling the placement problem in heterogeneous systems, where it is often not possible to place rules on certain nodes due to constraints, is not considered in existing work. As a result, todays placement solutions are not applicable in scenarios with many constraints.

Certainly, such constraints are of high relevance for real world deployments. CEP may comprise several heterogeneous CEP systems hosted with differing configurations at various domains (cf. [3]). As a consequence, the placement of rules is often restricted to a small subset of nodes, that have the corresponding capabilities and permissions. Moreover, CEP systems require a high level of decentralization since a node that gains global knowledge might be a security and scalability problem in heterogeneous networks. Furthermore, placement has to be adaptive, due to the dynamic behavior of CEP applications that stems from load variations, changes in the availability of correlation nodes as well as to changes in the rule deployment.

In this paper, we present a distributed solution to find and optimize valid placements in constraint-heavy distributed CEP systems, that has been implemented and tested within the DHEP framework [4]. Our optimization goal is to minimize the network usage of the running system. Our placement & optimization algorithm reacts dynamically on changes. As a result, the system can adapt autonomously to its environment. Our evaluations show that the algorithm finds near optimal placements even in very restrictive scenarios.

The rest of this paper is structured as follows. In Section II, we detail the challenges of placement in heterogeneous

CEP systems by means of the energy & utility example before presenting the system model and formal problem statement. In Section III related work is discussed. We present our approach in Section IV, which will be evaluated in Section V. Finally, we will have an outlook and conclusion in Section VI.

II. Problem Description

A. Motivation

Constraints and Resources: The high number of collaborations among business partners in todays world results in a cooperative nature of the involved companies business processes. As an effect, the processing and execution of business events is performed in heterogeneous environments, where nodes have different characteristics, resources, domains and processing capabilities. This imposes two main challenges that need to be tackled by placement solutions: a constraint-based handling of the functionality as well as the inclusion of resource usage. We exemplary show this with an energy and utilities scenario. Here, various energy providers and brokers as well as network hosts make use of event processing technology to enhance the efficiency of large power grids(c.f. Figure 1). They do so by exchanging information about energy provision and consumption within the system. The event processing is typically organized hierarchically among several substations, where information is processed and reacted on with the help of correlation engines. Smart meters are placed in consumers households and act as event sources and processing nodes at the same time: they emit events about current consumption on the one hand but also to steer energy consumption within the household on the other (e.g. based on energy prices).

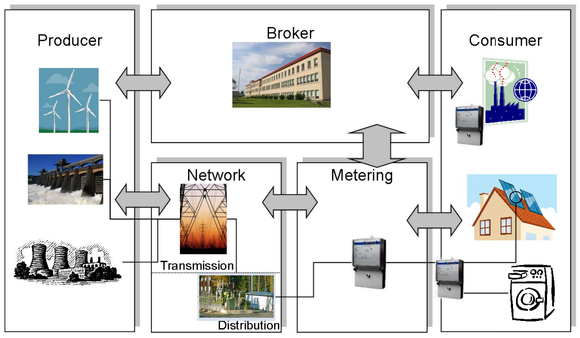


Fig. 1. Participants in an energy network

It can be seen, that different situations are detected at various domains (e.g. Broker, Consumer) within a very large system. Thus, event correlation is favored to be deployed in a distributed way. However, multiple constraints increase the complexity of the placement process:

(i) Heterogeneous Engines: The functionality needed at different levels within the power grid is of different complexity: while low-footprint event processing is sufficient in some places (e.g. filtering at power meters), major substations require more expressive processing capabilities. Therefore, different kinds of processing engines are installed within a large power grid (e.g. small engines at power meters, large powerful engines at substations). As a result, rules are not able to be processed at every node.

(ii) Domain Restrictions: Due to many participants in a large scale power grid (e.g. different energy providers), security becomes a major issue (e.g. confidentiality). Domain restrictions preclude the placement of rules on many nodes.

(iii) Heterogeneous Resources: The nodes within the network have differing resources. While energy providers are likely to possess large scaled dedicated servers with high-end computing capabilities, small power meter units at the customer are equipped with less memory and cpu power.

A placement algorithm which targets at finding solutions for this constraint-heavy scenario must be able to deal with these restrictions. This fact makes most placement algorithms used in CEP systems inappropriate, as they typically try to find the best placement (concerning a specific optimization goal) in the whole search space (cf. [12], [6], [16]), e.g. by introducing a latency space. However, constraint heavy applications typically require the search space to be pruned to a small subset of allowed placements. Consequently, a placement algorithm has to be able to search for solutions in the (most likely) small subset of the available nodes, depending on the given rule restrictions. Furthermore, since the solution space usually is very small, the algorithm should be complete, such that it can be guaranteed to find a solution if there exists one.

Moreover, most current placement algorithms do not consider the actual resource usage of rules on the nodes. However, the presence of a wide range of different machines with different capabilities, processing power or memory makes the introduction of resource usage necessary (cf. [4]). Nodes can be overloaded when *their* rules are causing a lot of computational effort, resulting in falsified and unreliable results. To the best of our knowledge, no algorithm exists that both considers constraints and resource consumption at the same time.

Dynamics: Despite handling constraints and resource requirements, an placement algorithm has to respect the dynamic behavior of CEP in heterogeneous environments:

(i) Rule Set: Although adding new rules is the typical example for dynamic behavior in CEP systems it is, in fact, rather uncommon. The reason is that CEP systems are usually not changed frequently in terms of rules. Once established, the set of rules is meant to run for a longer period and is going to report specific situations whenever they occur. Changes to the rule set often result from changes in the business logic of a company, for example when use cases are added or changed within a software. However, whenever a new rule is added, a quick deployment is desired.

(ii) Network: Resource overload on a node comes into account more often. It is usually a consequence of changing event rates, which result in a higher resource consumption of the affected rule(s). Once the available resources at a node get scarce, migration of one or more rules is necessary. Since rule sets are often based on real world events, the event rates are heavily dependent on uncontrollable sources and can therefore often not be predicted reliably.

As a consequence of the discussed dynamics we have to be able to react on changing conditions properly: i) solutions have to be found quickly to minimize the time a system is in an *invalid state*, i.e. a node is overloaded, or a rule restriction is not fulfilled. ii) due to changes in the system (which do not lead to an invalid state) a rather good solution may turn into a bad one over time. Consequently, the system needs to adapt itself in order to enhance the deployment during runtime.

B. System Model

We consider a set of interconnected and cooperating correlation nodes $N = \{n_1, n_2, ...\}$ that are connected via a P2P network and together host a user-specified CEP application. A CEP application is defined by a set of interdependent rules denoted by R and event streams E. Each rule $r \in R$ operates on a set of incoming event streams $e_{in}(r) \in E$ and creates outgoing event streams $e_{out}(r)$. A simple example of a typical CEP rule in our energy grid scenario is shown in Listing 1. Here, *powerConsumption* events from different households are aggregated and an *AggregatedPowerConsumption* event is forwarded for further processing.

```
rule AggregateCurrentConsumption
    WHEN SEQ(pc1, pc2 : PowerConsumption)
    IF pc1.meter ≠ pc2.meter
    RESTRICT engine.type = amit
    EMIT AggregatedPowerConsumption
                    (amount = pc1.amount+pc2.amount)
```

Listing 1. Example of a Simple Aggregation Rule

Both rules and nodes are dynamic: users may add, change and remove rules and nodes may join and leave the CEP system during runtime.

Rules can depend on each other. In particular, the outgoing event stream of a rule, $e_{out}(r_i)$, can serve as input to another rule, $e_{in}(r_j)$. Sticking with the given example, the *AggregatedPowerConsumption* events are used in another rule that determines energy overload within a sub network of the power grid. These interdependencies are described in a (directed, acyclic) rule graph $G = (R, E)$, where we create an edge $e_{i,j} = (r_i, r_j) \in E$ for every event stream between rules.

The set of nodes N can be heterogeneous with respect to the components and resources it provides. Resources, for example CPU or memory, are finite on every node and consumed by the rules placed on it. A component is defined as a non-resource attribute of a node that might be required by a rule: for example a specific CEP engine, or membership in a domain. Every node n_j has p components $\chi(n_j) = \{c_{j,1}, .., c_{j,p}\}$ and q resources $\gamma(n_j) = \{g_{j,1}, .., g_{j,q}\}$. An exceptional case of nodes are the sources and targets of events, that are not part of the processing network (i.e. they have no engine equipped and are therefore not capable of correlating events).

Rules can have different requirements with respect to resources and components. Every rule r_i has k component requirements $\alpha(r_i) = \{a_{i,1}, ..., a_{i,k}\}$ (which we call *constraints* in the following) and m resource requirements $\beta(r_i) = [b_{i,1}, ..., b_{i,m}]$. Additionally, we define the resource usage of an event stream $e_{i,j}$ as $\delta(e_{i,j}) = [d_{i,j,1}, ..., d_{i,j,s}]$. The distinction between resource usage of event streams and nodes is necessary, since the resource usage is heavily dependent on the incoming and outgoing event rates (cf. [4]).

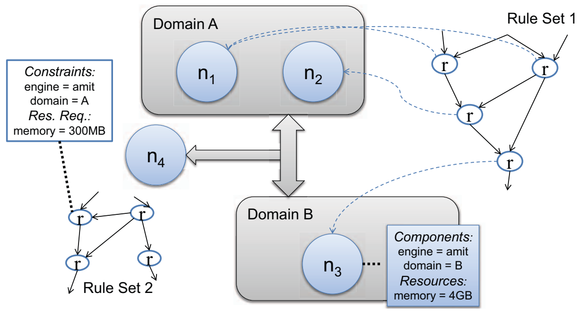


Fig. 2. Elements of the System Model

Within this system, we now try to find a valid placement of rules on nodes with respect to constraints and resource requirements (cf. Figure 2). Therefore, we define the boolean matching function $\mu(a, c) : \alpha \times \chi \to \{true, false\}$, which return *true* if the constraint $a_{i,k})$ is included in the set of components $\chi(n_j)$. A *valid placement* for a rule is given, if all constraints are fulfilled, i.e. $\forall a \forall c : \mu(a, c) = true$.

C. Problem Statement

Based on the given system model, we first formalize the general placement problem in distributed heterogeneous event processing networks (DHEP) and then state the optimization problem of minimizing network usage.

Definition 1 (The placement problem in DHEP): Given an event processing network N consisting of j nodes n and a set of i rules $r \in R$ that form a rule graph $G = (R, E)$, find a placement P with the mapping function $\pi(r) = n$, which assigns a rule to a node, such that:

$$\forall r : \mu(\alpha(r), \chi(\pi(r))) \quad (1)$$

$$\forall n, \forall c : \sum_{\pi(r)=n} (b_c) + \sum_{\pi(r_i)=n \vee \pi(r_j)=n} (d_{i,j,c}) \le g_c \quad (2)$$

whereas $\delta(e_{i,j}) = 0$, if $\pi(r_i) = \pi(r_j)$

Condition 1 ensures, that the constraints of the placed rules are fulfilled. Condition 2 ensures that the resource usages of the rules and event streams do not exceed the node and network resources. This means, that for all rules placed on a node n and all resources c the sum of these rules' resource requirements as well as the incoming and outgoing event streams of these rules is less or equal to the number of provided resources.

The formulated placement problem can be seen as a constraint satisfaction problem (CSP), where the rules constitute variables and the rule requirements are constraints that have to be matched by the machines. Based on this CSP, we can formulate the constraint optimization problem of minimizing network usage. Network usage is the bandwidth-delay product, which denotes the load that is on the network at a certain point in time. Hence, we formally define the network usage σ of an event stream as

$$\sigma(e_{i,j}) = delay(\pi(r_i), \pi(r_j)) \times dataRate(\pi(r_i), \pi(r_j)) \quad (3)$$

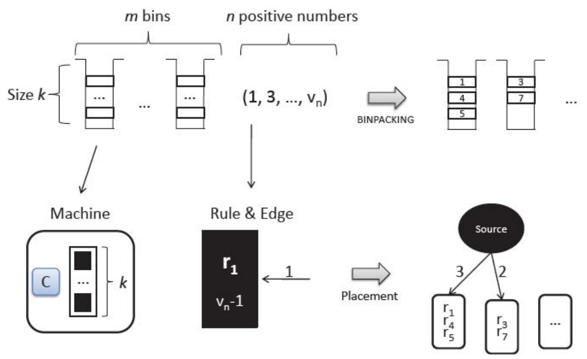


Fig. 3. Reduction to *Bin Packing*

Furthermore, we define the placement cost ϕ of a rule on a node as the sum of network usage of its incoming event streams, i.e. its incoming edges in the rule graph:

$$\phi(r, n) = \sum_{e_i \in e_{in}} \sigma(e_i) \tag{4}$$

With Equation 3 and 4, we define our goal:

Definition 2 (Minimizing Network Usage in DHEP): Given a cost function ϕ that determines the placement cost of a rule r on a node n. If $\mathcal{P} = \{P_1, ..., P_k\}$ is the set of all possible placements solving *the placement problem in DHEP* (c.f. Definition 1) and $\pi_{P_i}(r) = n$ is the assignent of rule r to node n done by P_i, $P_i \in \mathcal{P}$ is optimal iff for all $P_j \in \mathcal{P}$:

$$\sum_{Rules} \left(\phi(r, n)_{\pi_{P_i}(r)=n}\right) \leq \sum_{Rules} \left(\phi(r, n)_{\pi_{P_j}(r)=n}\right) \tag{5}$$

Note that by modeling also sources and targets in the rule graph and including only the incoming event streams, every edge is considered exactly once. Hence, a placement is optimal if it has the lowest placement costs in total. As a result, the communication load generated by the rule graph is minimized. Furthermore we would like to stress that the definition of our cost function could be easily extended with the resource usage of the nodes, as it is demonstrated in [4]. That means, we would not only consider the event traffic in the network, but also within the nodes. This would change the optimization goal to minimizing system usage.

Theorem 2.1: The placement problem in heterogeneous systems is NP-hard.

Proof sketch: Figure 3 illustrates how the problem can be reduced to *Bin Packing*. Here, nodes are mapped to bins, while rules are mapped to the values that going to be packed in the bins. The size of the bins is chosen by the free node resources, while the rules' resource consumption determines the values. Hence, if we could find an algorithm that solves the placement problem in deterministic polynomial time, the algorithm would also give a solution to pack the bins. □

III. Related Work

Placement in DCEP: With the increasing importance of distributed CEP, researchers have also tackled the placement of CEP functionality within a distributed CEP network. However, the existing solutions are unable to deal with heavy constraint settings, where only a small subset of nodes is appropriate for the placement of each rule. For example, [12] or [6] describe systems that enable distributed event composition where migration of rules is supported. However, the systems do not include any constraint respective behavior. Basically, placement is not restricted to a subset of nodes and also resource usage is no explicit variable. Other approaches, like [13] and [16] solve the multi-operator placement problem by means of spring relaxation techniques. A latency space is created in order to efficiently search for the best placement within the search space. However, it is not possible to restrict the search space based on constraints other than latency.

However, the CEP community has identified the need for placement algorithms that can handle rule restrictions. This is first mentioned in [7], where the authors present a description language where users can attach restrictions to rule definitions.

(Distributed) Constraint Optimization: Algorithms that do handle placement problems in high constraint settings can be found within the constraint optimization community where a lot of research has been done in the last decades (c.f. [8], [10]). This resulted in many distributed algorithms, that try to be both efficient in terms of completeness and speed (c.f. [9], [17]). However, these algorithms are designed for static problems, making it unefficent to deploy them in a dynamically changing distributed system like ours. For example, Branch-and-Bound optimizations [15], [14] have found a major consideration in the field of DCOP. While these methods are able to provide quality guarantees, they are mostly slow as their backtracking based algorithm relies on synchronous communication. Also, they are not able to optimize previous solutions. Instead, when changes to the system are made, the algorithms start all over again to find a new solution, which makes them unacceptable for our problem.

IV. Approach

Finding a close to optimal solution to the placement problem is expected to take significant processing time. This can lead to long phases in which the CEP system is inactive and events are unavailable. In order to ensure a high availability of the CEP system, our approach makes combined use of two algorithms. While the system is in an invalid state the algorithm aims to determine a *valid initial placement* for which all constraints are satisfied and no node suffers from overload. Here, we make use of heuristics that aim towards finding a valid solution early, yet still favor a better network utilization to get already good solutions considering our optimization goal. Once the initial solution is deployed, an *optimization* phase is started which aims at minimizing the network utilization.

During runtime, all correlation nodes operate without central coordination. Continuously, a node checks for all of its rules whether the constraints are satisfied. To ensure the availability of the system during optimization, we perform logical migrations: Every node creates an *update list* where it stores potential migrations for the rule it wants to get optimized. Actual changes to the system are made in a *commit phase*, where updates with respect to a rule deployment are executed.

The commit phase is entered, when the optimization algorithm has stopped. During the commit phase, every node that is participating in the running optimization deploys the new rules that are assigned to it.

Before presenting the details of the *initial placement algorithm* and the *optimization* phase in Section IV-B and IV-C, we first concentrate in Section IV-A on the common parts shared by the two algorithms.

A. Algorithm Basics

Our placement algorithm needs to react to the dynamics which stem from inserting new rules into the system or changing event rates. We therefore make use of a monitoring component at the node. If the changes result in an invalid placement, the affected node initiates a reconfiguration (i.e. *initial placement*). This may trigger subsequent migrations of correlation rules, if necessary, to find a valid placement. If the placement stays valid, the continuously running optimization algorithm will eventually search for new configurations that would result in a better cost. During this process, nodes go into a busy state while searching for alternative placements to prevent inconsistencies caused by parallelism. The generic course of action performed by a placement algorithm is characterized briefly in Algorithm 1. Both the initial placement

Algorithm 1 Generic Placement Algorithm

```
procedure MONITORCONSISTENCY
    while true do
        if (invalidState) then
            RECONFIGURE($r$)
        end if
        ENTEROPTIMIZATIONSTATE()
    end while
end procedure
```

Algorithm 2 Standard Routine of Placement Algorithms

```
procedure FINDPLACEMENT(r)
    IDENTIFY VALID CANDIDATE NODES($r$)
    RECEIVE CALCULATED DEPLOYMENT COSTS()
    CHOOSE TARGET NODE()
    MIGRATE($r$)
end procedure
```

and the optimization of a rule run through several typical steps when they are started. The steps are sketched in procedure *FindPlacement* in Algorithm 2. After the placement algorithms are started for a rule, we define our solution space by searching for nodes that fulfill the constraints, i.e. are in principle capable of hosting the rule. Then, each node checks whether it has enough free resources to deploy the rule and locally calculates the placement costs based on the expected network utilization (c.f. Section II-B). The result is sent back to the requesting node. Finally, among the replies a new node is chosen and the rule is migrated. Note that after migrating a rule, other subsequent migrations might be necessary to reach a valid state. Thus, we always perform these steps logically: the running system is only affected in the commit phase, where the changes are accepted and finally deployed on the nodes. Hence, optimization can run in parallel and is not continuously interrupting the CEP process.

Each of these typical steps will now be described in detail. However, although both the initial placement and the optimization algorithm follow this concept, they differ after the *identification of valid candidate nodes*. Therefore, we describe both mechanisms separately in Section IV-B and IV-C.

Identifying candidate Nodes: Candidate nodes are potential targets for a rule that needs to be migrated. Determining candidate nodes requires to deal with two difficulties: i) candidates must be determined fast and at low communication cost even though there is no central knowledge; ii) due to the presence of constraints the number of candidate nodes is typically restricted to a small subset of all available nodes. To overcome these difficulties, our candidate selection uses a content-based publish/subscribe middleware that is deployed as part of the CEP system (cf. [1]). Target nodes are determined by publishing the rule by means of its constraints. The key is, that each node subscribes to the constraints it is capable to suffice (cf. Figure 4). Currently, we specify the needed *engine type*, *domain* and connections to *database* as constraints.

The course of finding candidate nodes is depicted in Figure 4. When a rule must be placed within the system, the initiating node (n_4) publishes a deployment request with the rule's constraints. Publish/subscribe delivers the message to all subscribers: the nodes that can potentially deploy the rule. The deployment request contains two pieces of information: the rule and an identifier. The rule is essential for the cost calculation, the id is needed for handling parallelism in the system, since multiple requests can occur simultaneously.

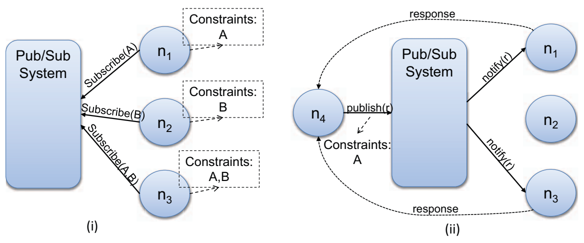
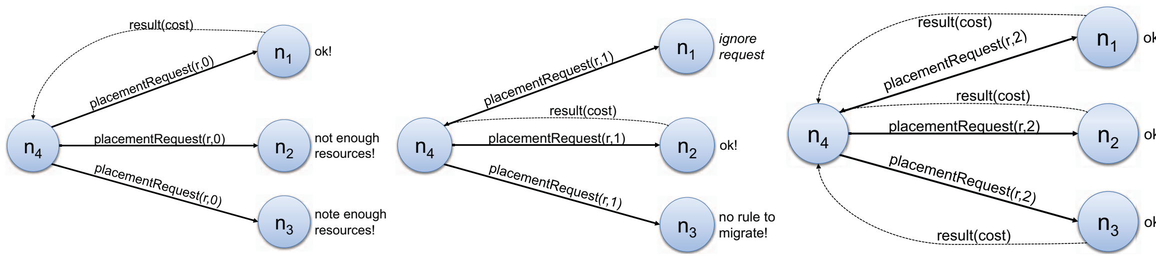


Fig. 4. Subscribing to Rule Constraints

B. Initial Placement

After receiving a placement request, every node first checks whether it can deploy the rule. Therefore, a node compares the estimated rule requirements with its own free resources. Basically there are two options: i) the node can deploy the rule; ii) the node cannot deploy the rule unless one or more rules are migrated. If the node can deploy the rule he sends an answer containing the cost for the rule deployment(i.e. the network utilization it would cause). However, when migration of other rules is necessary, the operation of placing a rule becomes complex. The reason is that these migrated rules might cause high overall costs on other nodes themselves after moving them, hence compensating for a good placement of the originally placed rule. Furthermore, the migrated rules might cause even more migrations subsequently.



(a) Priority0: Only Nodes with free Resources reply (b) Priority1: Nodes with easy migratable rules reply (c) Priority2: All nodes reply

Fig. 5. Requesting Placements with the 3-Way Heuristic

As a result, we use a 3-way heuristic which i) aims at reducing the number of migrations ii) is able to find placements with low costs and iii) guarantees to find a solution if there exists one. In our approach every request is associated with a priority. The larger the priority value the more nodes will answer a placement request. The algorithm uses three priority levels, and its behavior is sketched in Figure 5.

At priority level 0, only nodes that can directly deploy the rule reply to the deployment request, i.e. they fulfill all constraints and have sufficient resources. The reply contains the result of the cost function. In our example, the requesting node n_1 waits for the results, and choses the node which results in the best placement cost, i.e. the lowest network utilization value. If no node can be found that can directly place the rule, the request message is assigned priority level 1. Now, every node replies that can deploy the rule after migrating one or multiple other rules whose migration cost are below the cost of the rule. The migration cost is used in order to prevent rules from migrating that require a lot of migration effort. With this cost, we are able to order rules based on their migration effort. It is a fixed value assigned to each rule when it is defined and is typically defined by the amount of state information that has to be transferred in a migration process. Finally, priority level 2 is assigned to a request message if priority level 0 and priority level 1 messages did not succeed in finding a valid placement. Any node receiving a request message of priority 2 is answering the request, even if the migration cost of the already deployed rules is higher than that of the new rule. If there is no valid placement after priority 2 is reached, an error is sent to the node initiating the migration.

Replies to a deployment request are sorted by the initiator according to the deployment cost. For example, in Figure 5 the initial placement leads to a result list of n_2, n_4, n_3. Based on the result list, the requesting node sends a placement request to place the rule to the best node in the list. The recursive component of the algorithm comes into account at this point: if the node receiving the placement request can not deploy the rule without migrating one of its other rules (i.e. priority level 1 or higher), the placement process is initiated again. This procedure is depicted in Figure 6. In (a) Node n_2 is chosen to deploy rule r_n. However it is forced to migrate rules in order to do so. Therefore it creates a migration list containing the rules ordered by the migration cost. The node then tries to migrate rules based on the list, here r_1 and r_2. For both rules, a new placement procedure is initiated by n_2 subsequently in (b). If a placement fails, the next node in the result list is chosen (cf. r_2 in (b) and (c)). As can be seen in (c), node n_3 may also be forced to migrate one of its rules in order to find a valid placement. After a better valid placement is found, and the algorithm finally terminates, the nodes can update their state and deploy the new rules(d).

Backtracking: If a placement attempt failed and an error message is sent to the node initiating the placement request, the changes made during the placement attempt have to be revoked. Therefore, he sends a *rollback* message to every node participating in the placement attempt. The nodes can be identified by a *tracepath*-list held on the nodes. The rollback mechanism equates backtracking, i.e. jumping back to a valid state if an entered path does not lead to a solution.

Termination: To ensure termination of our algorithm, we have to avoid infinite cycles, which might occur in priority level 2, as well as deadlocks, which might occur when two independent placement algorithms are running in parallel. To avoid cycles, we add a *rulePath* variable to the placement request message. Every node receiving the request updates this variable by adding its own id. A cycle can now be detected whenever an identical sub-path in the *rulePath* is repeated. If a cycle is detected, we backtrack the request to the last valid state before entering the cycle. To avoid deadlocks, the requester waits for a random time until requesting again whenever a requested node is busy. If the node is still blocked, it has to cancel its running request if it has a lower id than the new request.

Properties: In this Section we discuss and prove properties of the algorithm. Lemma 4.1 states, that the algorithm is complete. Lemma 4.2 states, that the algorithm terminates fast if there exists a node that allows a valid placement and has enough resources available.

Lemma 4.1 (Completeness): Assume a stable system and the existence of a solution to the *the placement problem in DHEP* (c.f. Section II-C), where network conditions, nodes and rules do not change. The initial placement algorithm will eventually find a solution.

Proof sketch: Given the worst case in which finding a solution takes several migration steps. The placement algorithm will initiate a placement request at priority 2 and start backtracking over all possible placements. In particular it will find any existing solution. □

Lemma 4.2 (Fast termination): Assume a stable system state. If there exists both an invalid rule placement and a node with enough free resources to satisfy the constraints of this rule. A valid placement for the rule is found after the first iteration step.

Proof sketch: In the first iteration step, an event is published at priority level 0. The event contains the required resource types (as attributes). In a stable state all correlation nodes have subscribed to their resource types they are able to provide. Consequently, any node capable of satisfying the constraints of the invalidated rule will respond to the event and eventually deploy the rule. □

Further remark: If such a node does not exist, the algorithm will always need more than one iteration step, since (multiple) migrations are done. The time spent on finding a solution is heavily dependent on the length of the search path, and therefore on the number of migrations done. While the different priority levels are used to keep the search path as short as possible, the highest priority results in a lot of communication. Yet it is required for completeness, as any node is eventually considered in the placement process.

C. Optimization Phase

The previously presented initial placement task is solving the placement problem in heterogeneous environments, as it guarantees to find a valid placement if such exists. To find a near optimal solution, we optimize this initial placement in the optimization phase described in the following. On a time constant basis, every node tries to find alternative placements for rules, calculates the cost difference of the new placement and decides whether a migration should be initiated (as depicted in Figure 6).

Relating to the challenges stated in Section II-A, the algorithm for minimizing the network latency in a heterogeneous event processing system needs to fulfill a couple of major properties: i) decentralized coordination, and ii) avoid to be stuck in local minima.

Decentralized coordination prevents us from maintaining global knowledge and forces us to make decisions based on local knowledge. This is especially problematic when we determine the next rule that should be optimized, which should ideally be the *worst* placed rule in the system. To always choose the best result may lead to suboptimal solutions. The reason is, that the solution is found greedily, without being able to get back. As a consequence, the algorithm will run into a local minimum and may miss better solutions.

We overcome this issue by letting the nodes optimize *their* rules independently based on their local knowledge and applying techniques of simulated annealing. Our optimization algorithm is listed in Algorithm 3. After entering the optimization phase, every node periodically starts the optimization algorithm. At first, it chooses the *worst* placed rule that has not been optimized so far, i.e. the rule with the best optimization potential based on the node's local knowledge. It does so by comparing the cost of each rule with a *best known system cost* parameter that is distributed within the system during the optimization process. Thereafter, a migration probability is calculated which is determined by means of the current placement and the improvement potential (c.f. Section 4.1). The migration possibility increases with the improvement potential. This value is used to add fuzziness in the process of choosing the next rule that is going to be optimized. After a rule has been selected, placement options are searched and the best replying node is selected. The node is selected if either: i) the new placement cost is lower or ii) a randomly calculated number is higher than an acceptance function.

Algorithm 3 Pseudocode for the Optimization Algorithm

```
procedure INITLSA( )
    $r \leftarrow$ CHOOSE RULE FROM NODE( )
    if SEARCHALTERNATIVEPLACEMENTS($r$) then
        $diff \leftarrow$ CALCULATECOSTDIFF( )
        if MIGRATIONPROBAB($r$) $\geq RAND[0;1]$ then
            if $diff < 0$ or $e^{-\frac{diff}{T(n)}} \geq RAND(0..1)$ then
                UPDATENODES( )
            elseDISCARD( )
            end if
        end if
    end if
end procedure
```

It can be seen, that the basic principle of optimizing a current solution based on simulated annealing techniques shares similarities to a distributed greedy optimization: i) every node has limited knowledge that is based upon the *tracePath* of a placement request; ii) after predefined intervals, every node tries to find alternative placements for any of its rules. However, there exist some differences which are motivated by the major requirements to coordinate the optimization decentrally as well as avoiding local minima.

Properties: The presented algorithm allows to find near best solutions decentrally by adopting techniques from simulated annealing in a decentralized network and avoiding to get stuck in local minima. In the following, we present the algorithms properties and discuss their correctness informally.

Property 4.1 (Improvement potential): The optimization favors rules with a higher potential to improve the placement.

Discussion: as in classic simulated annealing, our optimization algorithm does not always migrate the rule(s) when finding an alternative solution. Instead, there exists a certain migration probability P which is determined by means of the achieved improvement and the best possible improvement Δ_{max}. The best possible improvement describes the maximum difference between all rule's worst and best placement. The migration probability increases, when the achieved improvement is closer to the best possible improvement. Formally, the probability is defined as:

$$P(r) = \frac{\phi(r, n_{current}) - \phi(r, n_{new})}{\Delta_{max}} \qquad (6)$$

However, in a distributed system Δ_{max} is unlikely to be known, since each node would have to know every possible placement. Thus, Δ_{max} is estimated locally. It is determined

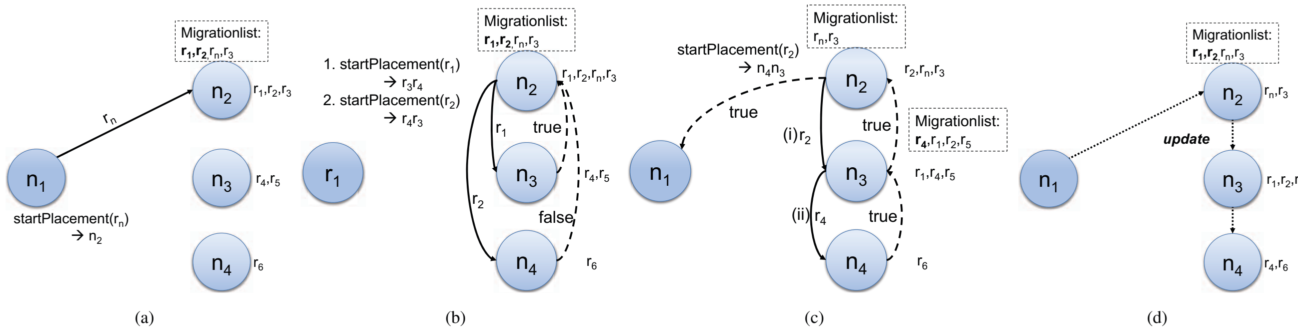

Migrationlist: r1,r2,rn,r3
startPlacement(rn) → n2
1. startPlacement(r1) → r3r4
2. startPlacement(r2) → r4r3
true
false
startPlacement(r2) → n4n3
Migrationlist: rn,r3
Migrationlist: r4,r1,r2,r5
(i) r2
(ii) r4
update
(a)
(b)
(c)
(d)

Fig. 6. Cost Calculation during Placement

by the maximum cost difference of all locally known rule costs. During the optimization process, various rule requests and replies reach the nodes and the values of Δ_{max} on each node adjust. In resource sparse systems, where many migrations are performed, the nodes finally have almost equal values for their local Δ_{max}. With that, a similar behavior like in the original simulated annealing algorithm is achieved, and the rules with the highest improvement potential are more likely to be chosen.

Property 4.2 (local minima): The optimization algorithm can avoid getting stuck in local minima, given a non-zero temperature value.

Discussion: as in classic simulated annealing, our optimization algorithm accepts a worse placement with a certain probability. This is done in order to be able to jump out of local minima, that would hinder a greedy algorithm from improving further. Therefore, we introduce an acceptance function that gives the algorithm a chance to accept worse placements too (c.f. Algorithm 3). The probability to accept a worse placement is dependent on two factors: the difference *diff* between the new solution and the current solution, as well as temperature value T, which is calculated by a temperature function. The probability to accept a worse placement increases with a higher T and a small *diff*. During the optimization process, the temperature function constantly reduces T to ensure the termination of the optimization process. Therefore, as long as T is non-zero, the algorithm may accept worse solutions and can escape local minima.

Property 4.3 (Termination): The optimization algorithm running at each node terminates, when one of two conditions is met: i) an optimization run has been started for each (local) rule or ii) a user-defined termination criteria is fulfilled.

Discussion: While the first condition ensures, that an optimization run is finished after every rule has been considered, the second condition is introduced in order to compensate a probably long and unwanted runtime. Because no heuristic is used to find quick solutions (as with the initial placement), it can occur that the chosen path in the solution tree is long. To overcome this problem, it is possible to add a user-defined termination criteria, which may for example stop the optimization run after a certain time or when the temperature value is below a certain threshold. Furthermore, the same mechanisms to avoid deadlocks and cycles are applied as in the *initial placement* algorithm (c.f. Section IV-B). After termination, the commit phase is started and the optimization results are deployed (if changes should be made). Note, that the optimization algorithm runs periodically, and there does not exist a final termination.

V. Evaluation

We implemented the presented approach within the DHEP framework [4] which enables complex event processing among heterogeneous nodes equipped with a variety of different event processing engines. Furthermore, it is possible to define restrictions on rules that need to be respected during the placement process. For example, rules can be restricted to run only in a specific domain, or require access to a specific database. As a consequence, there exist various rules that cannot be deployed on some of the nodes, but on others. On this basis we implemented the presented energy utility scenario from Section II-A. Here, *powerConsumption* events serve as the input of the CEP system. These events are filtered by (typically cheap and fast) different filter rules and aggregated by aggregation rules to *AggregatedPowerConsumption* events. Furthermore, more expensive sequence rules are used to calculate overload and underload situations. To all rules constraints were assigned concerning required engine types, domains and the connection to some specific database. At the same time, all our nodes are located in a domain, provide at least one engine, and may be equipped with a certain database.

For evaluation purposes, the presented setup was changed in size by adding additional rules and nodes. Furthermore, by increasing or reducing the number of existing domains or engines, the restrictiveness of a scenario could be changed.

Before discussing our evaluations in detail, we describe how the placement optimization is behaving during runtime. We show this by means of two variants of our approach: one with a greedy behavior (further mentioned as greedy algorithm) and one with a decreasing temperature value (further mentioned as simulated annealing-based algorithm). A greedy algorithm is achieved by only accepting better solutions, i.e. the temperature value is always 0. We compared to a centralized simulated annealing algorithm which makes use of its global knowledge to get better result with a higher probability.

We measured the overall placement quality after every optimization step by aggregating the cost of all placed rules (cf. Figure 7). Based on the result of the initial placement algorithm, the three techniques find improvements over time.

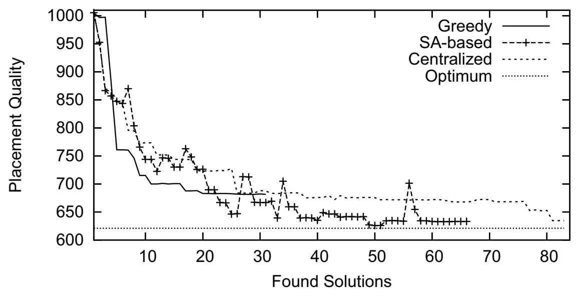


Fig. 7. Optimization Improvements over Time

In order to compare the solution's quality, we used a small scenario (28 rules placed on 12 nodes) to be able to compute the optimal solution. It can be seen, that the greedy optimization gets better continuously while the other optimizations also accept worse solutions with a certain probability. However, the greedy algorithm found a worse placement, while the others eventually get close to the optimum.

During long running applications, the behavior showed by Figure 7 is recurring: If the system changes the current placement gets worse (or even invalid). Consequently, the sytem will be adapted and a new placement is calculated/optimized.

A. Comparing different constraint levels

We evaluated the differences of the algorithm variants under varying different scenarios. We measured both the achieved quality and the caused traffic in terms of messages sent. For every scenario, we created multiple rule sets with varying constraints and deployed them on a network of 15 nodes. Exemplary we list the result of three evaluated variants in Table 1. Scenario 1 is a restrictive one. In average, about 2-3 nodes are possible placements for each rule in this scenario (~18%). In scenario 2 about 5 nodes can be used to place a rule (~35%). Finally, we created a constraint heavy scenario where only 1-2 nodes were able to place a rule (~10%).

Constraints:	high		medium		very high	
Algorithm	Quality	Msgs	Quality	Msgs	Quality	Msgs
Initial	997	9.2	877	13.1	1055	3.8
Greedy	923.8	17.2	777.8	29.8	1024.1	19.1
SA-based	867.4	24.8	769.6	34.5	1006.7	23.2
Optimum	789	-	721	-	949	-
Random	1158.3	9.2	1178.4	13.1	1219.3	13.1

Table 1. Optimization Results at different Scenarios

Table 1 lists the achieved algorithm quality and the messages sent for each migration in the different scenarios. That means, for migrating a rule during the initial placement, an average of 9.2 messages had been sent in scenario 1. It can be observed, that reducing the constraints of a scenario leads to a different result of the algorithms: On the one had, the overall quality that can be achieved is better, since there are more possibilities to find a better placement. On the other hand, this leads to a higher number of messages that are processed during the optimization process.

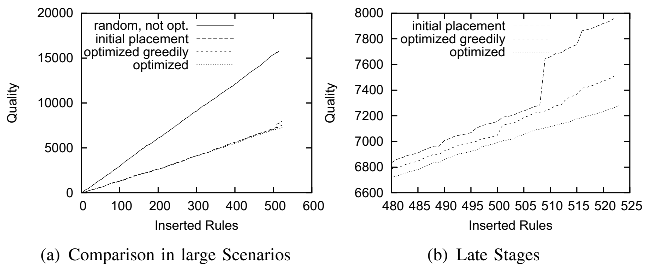
We can derive some important results. First, all optimizations resulted in more efficient placements. In our experiments we could reach a relative (to the optimium) improvement of 60-85% depending on the scenario. In scenario 2, the relative improvement was even more distinct than in scenario 1. However, as expected the initial placement's runtime (331ms in average) is significantly lower than for the subsequent optimization phase (9.5s for SA-based). Hence, the differentiation between the two algorithms is of high importance for the availability of the system. Second, the optimization algorithms achieve good results, compared to the best possible placement. In scenario 2 we were able to reach a result of 6.3% worse than the calculated optimal solution. Third, the more restrictive the scenarios are, the more they profit from our distributed simulated annealing algorithm: less messages are processed and the quality improvement is more promising. In scenarios that have less constraints like scenario 2, it is reasonable to favor a greedy algorithm. Here, the small difference between the optimization results does not justify the number of messages sent. The reason is, that the solution space in these scenarios is more flat, and therefore many local minima that are found by the greedy optimization are not much worse than the global optimum.

Finally, it is important to state that the traffic caused by our algorithm is negligible. Even in worst case scenarios, where many nodes are involved in the placement process, we processed in average 35 messages for migrating a rule. In a system, which processes hundreds and thousands of event messages per second, the communication overhead of the placement algorithm is marginal.

B. Insertion of rules in large Scenarios

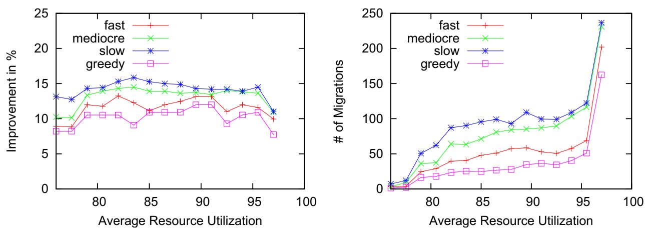
In our second evaluation, we created a network of 250 heterogeneous nodes. In this network, we continuously added rules, such that the system had to adapt itself on a constant basis. The rule input was created beforehand, to be able to reproduce the evaluation. Each rule had a randomized constraint. However, it was guaranteed that at least 10 different nodes exist matching this constraint(at least 4%). We added rules until our system was not able to place the next rule.

Figure 8(a) shows the achieved results of all algorithm variants. Several outcomes are apparent: First, as expected, the algorithms perform way better than a randomized rule deployment. Second, in the early and mid stages, where the system resources are sufficient, the difference between an optimized placement and an initial placement based on our 3-way heuristic is insignificant. Because there are enough free resources to place each rule directly on a good node without having to migrate other rules. Differences occur, as soon as



(a) Comparison in large Scenarios (b) Late Stages

Fig. 8. Quality under growing rule sets



(a) Improvement of initial Solution (b) Migrations per Optimization

Fig. 9. Behavior in restrictive Scenarios

some *good* nodes resources are used up and migrations might be needed to achieve a valid placement. This behavior can be seen in the late stages of the experiment, as depicted in Figure 8(b). Based on our experiment, the use of the SA-based algorithm promises the best solution, however seems to be most efficient at scenarios where resources get scarce.

C. Concerning high Resource Requirements

In this evaluation we compare the behavior of our algorithm based on three different temperature values: One *fast* converging temperature value, causing the algorithm to behave greedily soon; one *slow* converging temperature, allowing the algorithm to avoid local minima for a longer time; and one *mediocre* temperature in between. We compare the three algorithms with the greedy algorithm under a changing amount of available resources. By reducing the available resources, the overall resource utilization is increasing. This means, we increase the restrictiveness of the scenario stepwise, hence reducing the solution space of the placement problem and making it more difficult to find a valid placement.

Figure 9(a) shows the quality improvement achieved by all four optimization strategies relative to the scenario restrictiveness. Figure 9(b) shows the number of migrations needed during optimization to find the results relative to the restrictiveness of the scenario. In the depicted evaluation, the last valid placement could be found at an average resource usage of 97%. There were no valid solutions after reducing the resources even further.

Remarkably, it can be observed that finding valid solutions for highly restrictive scenarios (i.e. where almost all resources are consumed), results in more migrations (and therefore in a higher runtime). The reason is, that for optimizing the placement of a rule, it is more likely that another rule has to be migrated subsequently. Hence, the optimization under extreme resource requirements (>95%) resulted in a high number of migrations. This is especially true with slow converging temperature values, where additional migrations are performed due to accepting worse placements with a higher probability. This behavior can also be observed on other scenarios. As a consequence, we conclude: i) independent of the resource conditions on the nodes there exists a tradeoff between the effort one is willing to put into optimizing the solution, and the quality of the result he will get; ii) Under extreme conditions, where the whole systems suffers from extreme resource consumptions, our optimization strategies may require many migrations to eventually find valid solutions. This will highly increase the runtime of the algorithms. Here, we propose to prefer a greedy optimization to keep the runtime acceptable. However, we believe that these extreme conditions are rather unusual and should not happen regularly.

VI. Conclusion

This paper has addressed the placement of rules in heterogeneous event processing systems where it is important to cope efficiently with many constraints. The algorithm can adapt to dynamics that stem from load variations as well as to changes in the rule deployment. The analysis and evaluations show that the approach is particular beneficial in heavy constraint settings. Here, it finds near optimal placements with low cost by efficiently restricting the search space of possible placements. Furthermore, the separation in two phases – the initial placement phase and the subsequent optimization phase – contributes to a fast operation of the event processing which is crucial in the practical operation of event processing networks. In our evaluations this was true until we reached at an average resource consumption of ~95%. Future work will concentrate on the selection of the temperature function and a more efficient use of publish/subscribe to significantly reduce the communication cost in low constraint scenarios.

Acknowledgement

This work has been supported by the Baden-Württemberg Stiftung as part of the "CEP in the Large" project.

References

[1] A. Tariq et al. Dynamic publish/subscribe to meet subscriber-defined delay and bandwidth constraints. In *Proc of the Int. Conf. on Parallel Computing (EURO-PAR)*, 2010.

[2] A. Adi and O. Etzion. Amit - the situation manager. *The VLDB Journal*, 13(2), 2004.

[3] B. Schilling et al. Event correlation in heterogeneous environments. *Information Technology*, 5, 2009.

[4] B. Schilling et al. Distributed heterogeneous event processing: Enhancing scalability and interoperability of cep in an industrial context. In *Proc. of the 4th Int. Conf. on Distributed Event-Based Systems*, 2010.

[5] S. Chakravarthy and D. Mishra. Snoop: an expressive event specification language for active databases. *Data Knowl. Eng.*, 14(1), 1994.

[6] E. Fidler. *PADRES: A Distributed Content-Based Publish/Subscribe System*. PhD thesis, University of Toronto, 2006.

[7] G. Koch et al. Cordies: Expressive event correlation in distributed systems. In *Proc. of Conf. on Distributed Event-Based Systems*, 2010.

[8] F. Glover and M. Laguna. Tabu search. *Kluwer Academic*, 1997.

[9] K. Krishna et al. Distributed simulated annealing algorithms for job shop scheduling. *Systems, Man and Cybernetics, IEEE Transactions*, 25:1102–1109, 1995.

[10] S. Kirkpatrick, C. G. Jr, and M. Vecchi. Optimization by simulated annealing. *SCIENCE*, 220:671–680, 1983.

[11] G. Li and H.-A. Jacobsen. Composite subscriptions in content-based publish/subscribe systems. In *Proc of the Int. Middleware Conf.*, 2005.

[12] P. Pietzuch et al. A framework for event composition in distributed systems. In *Proc. of the 4th Int. Conf. on Middleware (MW'03)*, 2003.

[13] P. Pietzuch et al. Network-aware operator placement for stream-processing systems. In *Proc. of the Conf. on Data Engineering*, 2006.

[14] P.J. Modi et al. Adopt: asynchronous distributed constraint optimization with quality guarantees. *Artif. Intell.*, 2005.

[15] R. Ezzahir et al. Dynamic backtracking for distributed constraint optimization. In *Proc. of EVAI*, 2008.

[16] S. Rizou et al. Solving the multi-operator placement problem in large-scale operator networks. In *Proc. of 19th Int. Conf. on Comp. Comm. Networks*, 2010.

[17] M. Yokoo and K. Hirayama. Algorithms for distributed constraint satisfaction: A review. *Autonomous Agents and MA Systems*, 3(2), 2000.