MigCEP: Operator Migration for Mobility Driven Distributed Complex Event Processing

Beate Ottenwälder, Boris Koldehofe, Kurt Rothermel
Institute of Parallel and Distributed Systems
University of Stuttgart, Stuttgart, Germany
{beate.ottenwaelder,boris.koldehofe, kurt.rothermel}@ipvs.uni-stuttgart.de

Umakishore Ramachandran
College of Computing
Georgia Institute of Technology,
Atlanta, GA, USA
rama@cc.gatech.edu

ABSTRACT

A recent trend in communication networks — sometimes referred to as fog computing — offers to execute computational tasks close to the access points of the networks. This enables real-time applications, like mobile Complex Event Processing (CEP), to significantly reduce end-to-end latencies and bandwidth usage. Most work studying the placement of operators in such an environment completely disregards the migration costs. However, the mobility of users requires frequent migration of operators, together with possibly large state information, to meet latency restrictions and save bandwidth in the infrastructure.

This paper presents a placement and migration method for providers of infrastructures that incorporate cloud and fog resources. It ensures application-defined end-to-end latency restrictions and reduces the network utilization by planning the migration ahead of time. Furthermore, we present how the application knowledge of the CEP system can be used to improve current live migration techniques for Virtual Machines (VMs) to reduce the required bandwidth during the migration. Our evaluations show that we safe up to 49% of the network utilization with perfect knowledge about a users mobility pattern and up to 27% of the network utilization when considering the uncertainty of those patterns.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Distributed networks; C.2.4 [**Distributed Systems**]: Distributed applications; E.1 [**Data Structures**]: Graphs and networks

Keywords

mobility; complex event processing; migration

1. INTRODUCTION

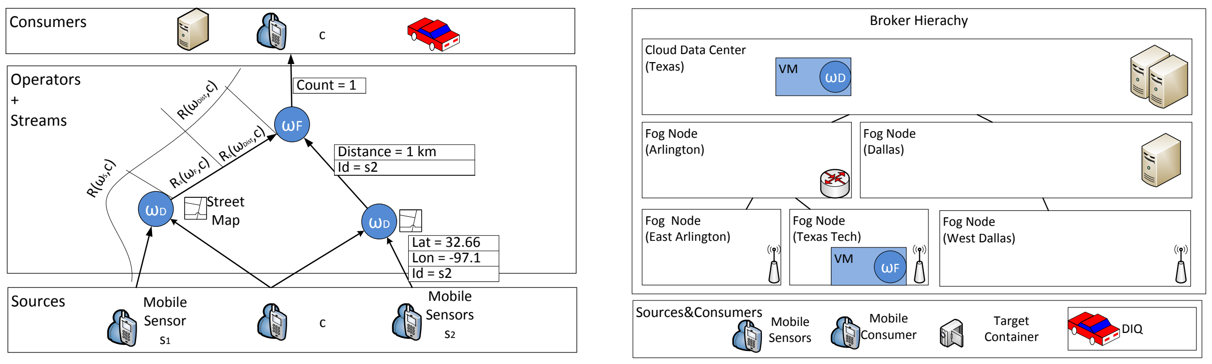
Over the last decade, the deployment of powerful mobile sensors and large scale sensor networks, monitoring mobile objects or locations, increased tremendously. Examples are off-the-shelf smartphones [5] and CCTV camera networks [25]. They enable novel real-time applications, such as continuous video monitoring of suspects [13], or traffic information systems [18], which provide information of interest to consumers.

Complex Event Processing (CEP) is a key paradigm to realize such applications. Changes in sensor measurements are modeled as events, while the application is modeled as set of event-driven operators. Such operators take streams of events as input, process them and produce new event streams.

Virtualized computing environments, i.e., clouds or fogs [3], provide elastic resources, which is highly appealing to support large-scale CEP systems. Cloud data-centers offer virtually endless resources to execute a vast amount of operators, however, impose a high communication latency since it requires transfering events from a user through the core network to the data center. Fog-computing, a resource paradigm proposed by Cisco [3], allows for processing on resource-constrained devices near users, like routers, for low end-to-end latencies. A federation of both clouds and fogs can support highly heterogeneous systems, where network-intensive operators are placed on distributed fog nodes and computational-intensive operators in the cloud.

In *Distributed CEP (DCEP)* [21], research [20, 22] has shown that the placement of operators has a significant impact on important performance metrics, such as network utilization and end-to-end latency. Furthermore, in *Mobile CEP (MCEP)* systems, where consumers and sensors are mobile, bandwidth and latency of streams are expected to change frequently with continuous location updates. For example, the number of available camera streams or the latency between the access point of a mobile sensor and the fog node where an operator is placed varies. To ensure the system's performance it has to constantly adapt the placement through migrations to new fog nodes.

However, each migration comes with a cost because operators are associated with local state, e.g., a cached portion of the event stream or a street map. This state can accumulate up to several GB of data [24] that have to be transferred during a migration. Frequently performing migrations to find better placements thus can significantly decrease system performance. For example, migrating GBs of state with each cell change of a consumer in a GSM network, while only several MBs of data are streamed to and from operators increases the network utilization.



(a) Operator Graph (b) Broker Hierarchy

Figure 1: System Model and CEP Operator Graph

In this paper, we propose methods for providers of virtualized environments to support operator migrations in MCEP systems. These methods exploit application knowledge of the MCEP system and predicted mobility patterns to plan the migration ahead of time. First, it allows us to amortize the migration costs by selecting migration targets that ensure a low expected network utilization for a sufficiently long time. Second, it allows us to serialize the operator for the migration and migrating parts of the operator a priori in away where unnecessary events are not migrated and bandwidth is reduced. In more detail, our contributions are:

1. The definition of a probabilistic data structure, called *Migration Plan (plan)*, which defines future targets and times for the migration, and a distributed algorithm to create such plans.
2. A migration algorithm, which uses a plan to minimize the network utilization while keeping the end-to-end latency below a threshold.
3. An analysis and evaluation study of the cost imposed by the creation and execution of a plan and its benefits.

The remainder of the paper is structured as follows: Section 2 introduces the underlying system model and Section 3 clarifies the problem. We present in Section 4 our plan-based migration approach. In Section 5 we present findings from our analysis. The approach's evaluation is presented in Section 6. Section 7 discusses related work before we conclude the paper and give an outlook on future work in Section 8.

2. SYSTEM MODEL

2.1 MCEP model

The operation of a MCEP system is commonly modeled by a directed, acyclic *operator graph* $G = (\Omega \cup S \cup U, L)$ where Ω denotes the set of operators, S the set of sources (e.g., range queries [18] or a mobile sensor in a smartphone), U the set of consumers, $L \subseteq (\Omega \cup S \times \Omega \cup U)$ the event streams between operators, sources, and consumers. For example, Figure 1(a) depicts an operator graph that can detect the number of friends that were closer than 1 km to a user c over the last hour. Primary events are location updates of users, associated with a source-specific id, which are streamed to ω_D. Operator ω_D utilizes a street map to compute the distance between a friend and c. Operator ω_F then counts the number of distances smaller than $1km$ with disjoint source-ids and publishes the result on a social platform or the mobile device of the consumer. More sophisticated operator graphs can report about dangerous traffic situations [18] or allow monitoring of suspects [13].

Each $\omega \in \Omega$ encapsulates an arbitrary function that allows detecting patterns on deterministically ordered event streams. Events of those streams are managed in a set of dedicated queues Q. The number of buffered events in Q depends on the operators' semantics, which determine the selection and consumption of events from the queues [6, 21].

We refer to these dynamically changing queues as *mutable state*, while the *immutable state* of an operator is the part of the operator that is read-only and fixed in size. For example, the map of ω_D, a database for face recognition, or the executable code of the operator. Intermediate results, i.e., states of variables in an operator's implementation, are denoted as *computational state*.

2.2 Broker Hierarchy

The operator graph is deployed on a federation of k distributed brokers $\{b_1 \dots b_k\}$. Similar to typical mobile infrastructures, e.g., GSM networks or location services [9], brokers are organized in a spatially-partitioned hierarchy. The communication delay to mobile consumers decreases in a root to leaf direction.

The broker hierarchy is implemented through a combination of cloud data-centers and fog nodes (see Figure 1(b)). The latter can be routers or nearby workstations, that provide virtualized resources[*]. Such a hierarchy roughly approximates the network topology, since core network devices on the path from the consumer to the data center can be utilized for the processing. Within this mobile infrastructure each $\omega \in \Omega$ is hosted by its own dedicated VM that provides the execution environment for the operator.

Mobile consumer and sources share event streams over a wireless interface with the broker hierarchy. In order to improve their expected link quality, they greedily connect to the topologically closest broker, denoted as leaf broker. Since processing tasks are deemed to consume a lot of energy, those mobile devices are only thin clients, leaving all the processing to the infrastructure.

[*]Such in-network virtualization, i.e., fog computing [3], is currently promoted by companies like Cisco.

3. PROBLEM DESCRIPTION

A placement $P_{ts,te}$ is the assignment of operators $\omega \in \Omega$, of a given operator-graph G, to brokers in between time t_s to time t_e. When a mobile source or consumer connects via new access-points to new leaf brokers the end-to-end latencies and the network utilization can increase since now more brokers are potentially involved in transferring event streams. For example, when the source in Figure 2 changes its connection from the leaf broker b_1 to b_2, it also means that from now on all streams for ω_D, placed on b_1, have to be transferred from b_2 to b_1, possibly over multiple hops in the broker hierarchy. In order to improve the system performance, the system has to adapt the placement $P_i = P_{ts,te}$ at time t_e to a new placement P_{i+1} via migrations $M_j, .., M_k$ of one or multiple operators. To reduce the network utilization imposed by keeping ω_D at b_1 the system migrates ω_D to b_2, which can also affect the placement of ω_F.

Note, that also migrations themselves may impose a significant cost in terms of network utilization, given that a migration requires transferring the whole state of an operator to a new migration target which can be as large as several GB. Therefore, it is important to account for those costs when performing a migration. Instead of always deploying the best possible placement, a placement should only be deployed if its migration costs can be amortized by the gain of the next placement. This gain depends on many dynamic parameters such as the mobility of sensors and consumers as well as the actual workload of the event processing system.

The decision to initiate a migration may also be unavoidable in some cases for ensuring the responsiveness of the CEP system, e.g., to allow users to respond to traffic situations in real-time. In this case, the global end-to-end latency has to stay, at least on average, below a consumer-defined restriction R_c. In particular, it has to stay below the restriction on all paths in G from any source $s_i \in S$ to a consumer $c \in U$. On these paths, the detection is delayed due to communication delays between neighboring operators and the computational delay, the time that an operator requires to process a new input event. To ensure that the global end-to-end latency stays below R_c, the sum of the computational delay $d_c(\omega)$ and in- and outgoing streaming delay $d_{in/out}(\omega)$ of all those coordinated ω has to stay below a local $R_{(\omega,c)}$. For instance, in Figure 1(a) ω_D and ω_F reserve a part of their restriction for the global optimization.

When preserving latency constraints it is also important to consider the downtime of operators during migrations, as a consequence of having to halt an operator and start it again at the migration target. When all state is already available on the migration target at the migration time we achieve a minimal downtime, since the operator can start processing immediately. Consider, for example, in Figure 2, the map and event streams can already be transferred to b_2 before the source connects to b_2 and the operator ω_D has to be migrated to b_2. However, uncertain future locations of the source may lead to a situation where the network utilization is increased because the state is copied to b_2 but the source never connects to b_2 and a migration is unnecessary. This requires to migrate the operator and its state to future placements where the expected network utilization is low.

The *mobile migration problem* addressed in this paper is to find a sequence of placements $P_1, P_2, ..., P_p$ and Migrations $M_1, M_2, ..., M_m$ for an operator graph G in a broker hierarchy, where the overall network utilization is low and the consumer defined latency restriction is met.

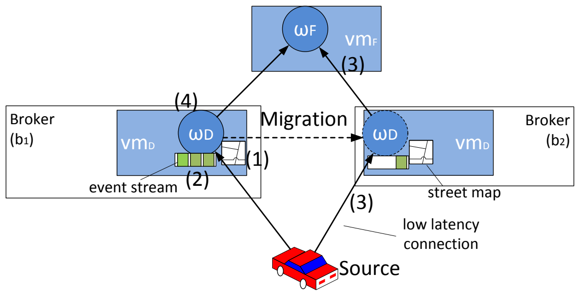


Figure 2: Migration

More formally, we consider the average bandwidth-delay product during a time interval as the metric to express network utilization. Let $bdp(P_i)$ be the sum over the average bandwidth-delay product of links in the broker network in a placement P_i. Then, the cost for a placement in between time t_s and t_e is $C_{pla}(P_i) = bdp(P_i) * (t_e - t_s)$. We can define the costs $C_{Mig}(M_j)$ for each migration M_j, accordingly. Furthermore, let $d(s_i, c)$ be the end-to-end latency from any source $s_i \in S$ to any consumer $c \in U$ over a path $(\omega_1, \omega_2, ..., \omega_n)$ in the operator-graph.

An optimal sequence of migrations is found if:

1) $C_{tot} = \sum_{i=0}^{p} C_{pla}(P_i) + \sum_{i=0}^{m} C_{Mig}(M_i)$ is minimal

subject to

2) $\forall c \in U : \sum_{i=0}^{|S|} d(s_i, c) \leq R_c$ at all time

4. PLAN-BASED MIGRATION APPROACH

The MCEP live migration system assigns a dynamically updated live *Migration Plan (plan)* to each operator. The plan defines for an operator a set of future migration targets, a deadline by when an operator needs to be ready to execute at its migration target, and a time when the migration will be initiated such that the migration deadline can be met.

At the time when according to the plan a migration needs to be initiated for an operator, the live migration system will begin to copy first the execution environment (VM) and immutable state (step (1) in Figure 2), then relevant mutable state that is required for future selections when the operator starts executing at the migration target (e.g., only one of three possible events in step (2)). Since the latter state allows to recompute computational state at the target, we don't need to transfer computational state. In the meantime the operator will continue to execute at its original placement until the transfer of state has been completed. Finally, at the times indicated in the plan, events are streamed to the new placement (step (3)) and the resources taken up by an operator at its original location are released (step (4)).

The performance characteristics of the live migration depend on how plans are created, as well as how they are adapted to the dynamics of the MCEP systems, i.e., where sources and consumers are connected to the broker hierarchy, how the usage of resources changes as well as how the load of the event system varies.

In this section, we first propose a simple model for a plan that triggers migrations at discrete time steps (see Subsection 4.1) and later show how to coordinate an operator's plan with the plans of its neighbors. Furthermore, we will show

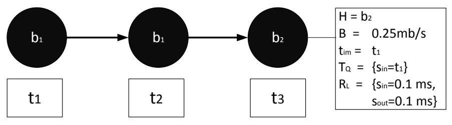


Figure 3: Basic Migration Plan

how to refine the basic mechanisms to deal with the inherent uncertainties which follow from the mobility of sources and consumers, variations in the workload, and resource usage in the broker hierarchy (see Subsection 4.2).

4.1 Creating a Migration Plan

The live migration system continuously anticipates the movement of its connected sources and consumers, as well as the load of event streams and latencies between the operators connected in the MCEP system in order to determine a plan. For mobile sources and consumers, this is achieved by dead reckoning mechanisms [28] or relying on information from navigation systems. The event load on links is estimated on average over the most recent traffic measurements, while latencies can be estimated via regular ping messages between neighbors or using Vivaldi Coordinates [8].

The quality of a created plan depends on the accuracy of this information; in fact unforeseen behavior may degrade the performance properties of our system and in the worst case even violate constraints. For now, however, to understand the basic principle of the proposed migration scheme we will treat this information as if it was accurate.

In this context we will answer

- How far and at which granularity do we have to plan migrations ahead?
- What are possible migration targets that can be incorporated in the migration plan of a single operator?
- How should the migrations of multiple operators be best coordinated?

We address this by first proposing the basic Migration Plan model, which allows studying migrations at varying granularities. In order to select possible migration targets, we propose the *time-graph model*, that allows finding migration targets, depending on the plans of other operators. The proposed coordination algorithm negotiates the plans with dependent operators and migration targets, in order to find minimum cost plans for each operator and reserve resources to ensure the execution of an operator at all times.

Migration Plan Model

A plan describes the upcoming migration behavior of an operator ω, determining when it is going to be hosted at which broker, as well as its expected resource requirements. In other words, it is the trajectory of an operator in the broker hierarchy. As depicted in Figure 3 for ω_D of the example, the plan provides an expected placement of an operator for a sequence of discrete time steps $t_i \in TS = \{t_0, ..., t_{max}\}$. An expected placement is a 5-tuple $ep = (H, B, t_{im}, T_Q, R_L)$, where H is the expected host of the operator, B the average expected bandwidth of the operator's outgoing stream, t_{im} is the time, at which the transfer of immutable state has to be initiated, T_Q a deterministic starting point for each queue $q \in Q$ from which events on the mutable state has to be transferred, and R_L indicates which part of the latency restriction is reserved for the incoming and outgoing streams L of ω.

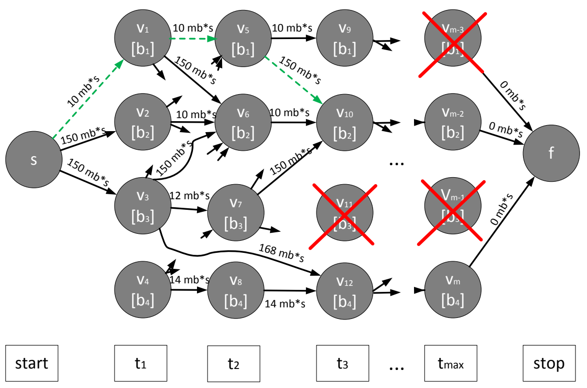


Figure 4: Timegraph Example

Since consumers and sources are constantly connected to leaf brokers, we can use the same abstraction to model and share their movements. For example, the plan indicates for a mobile sensor to which leaf broker it will be connected to and the expected size of its sensor stream.

Time Graph

The *time-graph* $= \{V_{tg}, E_{tg}\}$ (see Figure 4) is a data-structure that allows for each individual operator ω to identify costs and durations of future migrations and placements. The time-graph for a single $\omega \in \Omega$ comprises migration targets which are suitable to fulfill the constraints of a placement. Note, that even not performing a migration imposes a cost by streaming events over the in- and outgoing streams.

Each vertex $v_{tg} \in V_{tg}$ represents the possible placement of ω at a broker b, starting at time step $t_i \in TS$ and ending at the next time step $t_{i+1} \in TS$. For example vertex v_1 in Figure 4 is labeled with b_1 at t_1, which represents the placement at broker b_1 starting at t_1. Two special vertices which do not indicate future placements, labeled as s and f in Figure 4, represent the start and end of all possible sequences of migrations and placements.

Each directed edge $e_{tg} = (v_j, v_k) \in E_{tg}$ represents the migration between brokers or no migration if e_{tg} connects the same broker. Furthermore, an edge indicates that a migration starting at time step t_j of v_j is expected to be completed at time step t_k of v_k. For example, when the migration from b_3 to b_4 is started at time t_1, then all state is expected to be transferred by t_3 and ω can start processing.

The edge weight $w_{j,k}$ of $(v_j, v_k) \in E_{tg}$ represents the costs that are expected to occur in the time interval $[t_j, t_k)$ – the average bandwidth-delay products weighted with that interval (see Section 3). This comprises the expected migration costs $C_{Mig}(v_j, v_k)$ and expected placement costs $C_{pla}(v_{tg})$ at the broker of v_j during $[t_j, t_k)$. Note that the placement costs are not only considered in the case that an edge represents no migration but also if it represents a migration. This is because ω still processes events at the previous broker, while the migration happens in the background. The edge weight of $10mb * s$ between two subsequent vertices of b_1 are the placement costs during that time step, while the edge weight of $168mb * s$ between v_3 and v_{12} comprises the placement costs of $2 * 12mb * s$ at b_3 and migration costs of $144mb * s$.

```
 1: $stablePlan, tmpPlan, neighborPlans, TG \leftarrow \emptyset$
 2: upon trigger generatePlan( )
 3:   if notInAwaitState() then
 4:       await notWaitForFeedback() $\wedge$ noTargetCoordination()
 5:       $TG \leftarrow$ createTimeGraph($TG$)
 6:       $path \leftarrow$ determineShortestPath($TG$)
 7:       $P \leftarrow$ createInitialPlan($path$)
 8:       if P deviates from tmpPlan then
 9:           $tmpPlan \leftarrow P$
10:           trigger send $planUpdatedMsg(P)$ to all neighbors
11:       else
12:           tryInitTargetCoordination()
13:       end if
14:   end if
15: end
16: upon receive planUpdatedMsg($plan$) from neighbor n
17:   $neighborPlans[n] \leftarrow plan$
18:   trigger $generatePlan()$
19: end
20: upon receive planAcceptedMsg()
21:   tryInitTargetCoordination()
22: end
23: function tryInitTargetCoordination()
24:   if allFeedbackReceived() $\wedge$ noPlanGeneration() then
25:       checkNeedSendPlanAcceptedMsg()
26:       checkNeedSendReservationRequest(tmpPlan)
27:   end if
28: end
29: upon receive reservationReplyFromTargets($tmpPlan$)
30:   if tmpPlan is executable on all targets then
31:       $stablePlan \leftarrow tmpPlan$
32:   else
33:       trigger generatePlan()
34:   end if
35: end
36: upon timeout($timer\ in\ regular\ intervals$)
37:   if checkMonitoring() $\vee$ checkLatencyRrestriction() then
38:       trigger $generatePlan()$
39:   end if
40: end
```

Figure 5: Generating a Migration Plan

Migration Plan Creation

We now detail a distributed algorithm for the creation of the basic plan for an individual operator ω (see Algorithm 5), executed at the operator's current host. The algorithm finds a shortest weighted path in a time-graph (TG) that is local to ω and generates an executable plan ($stablePlan$) from this path. The time-graph is maintained using the set of plans from all neighbors in the operator graph ($neighborPlans$) and the anticipated event loads and restrictions given by the expected placements in these plans.

The first plan is created after the deployment of an operator. Afterwards, the plan creation is triggered concurrently when the latency restrictions are no longer preserved on the current broker, predictions on the outgoing bandwidth and collected latencies deviate beyond a threshold (Line 36-40) from previous predictions, or the plan of the neighboring operators changes (Line 16-19).

In the first step of the plan generation the *time-graph* is created, which models the expected placement and migration costs for this operator for the next TS time steps (Line 5). In a subsequent step an initial plan is determined by traversing the shortest path in the time-graph. This path contains a sequence of migration targets and times to migrate between them, where the expected overall network utilization is low and the latency restrictions are expected to be preserved (Line 6-7). We replace a previous stable plan with the newly found plan only if this new plan deviates in the expected placements at any time step (Line 8). However, before implementing the new plan it is consolidated by coordinating it with the neighboring operators (Line 10) and the brokers that are selected as migration targets. Migration targets might not have enough resources to execute the plan or neighbors adapt their own plan due to the plan that is to be consolidated. In both cases another iteration of the plan generation is triggered (Line 18/33).

Maintaining the time-graph in a stable manner. Algorithm 6 outlines the creation of the time-graph TG. At first a set of brokers that have in general enough resources to execute the operator are selected as possible future migration targets (Line 2). Then it populates an empty graph with the start and a stop vertex (Line 3).

In the following steps vertices for each combination of discrete time steps *TS*, starting at the current time *now*, and possible migration targets (*Targets*) are created (Line 4-5). However, vertices are omitted when the associated broker comes not into consideration to host an operator at $ts + now$ (e.g., v_{11} in Figure 4). This is the case when latency restrictions are not expected to be preserved or a local information on the migration targets, resource reservations indicates that not enough resources to execute the operator at the potential migration target are available.

When the vertex v is created, the algorithm anticipates which migrations are possible and which previously created vertices are therefore connected to v over a directed edge (Line 7-14). This requires to estimate the size of the mutable state, according to the average size of the queues and the time t_{mig} that it takes to migrate all state between any possible host b_p and the host b of v. Since t_{mig} may span over more than one time step, the vertex that represents b_p at time step $now + ts - t_{mig}$ is selected as source of the edge e. However, the operator has to continue its execution at b_p during the the migration, which means that all vertices labeled with b_p between $ts + now - t_{mig}$ and $ts + now$ have to exist (Line 10).

The weight of this new edge is the sum of the placement costs $C_{pla}(v)$ of all vertices representing b_p — the sum over the average bandwidth-delay product between the b_p and the expected placement from the neighbors plan weighted with the time intervals during the given time steps — and *migration cost* $C_{Mig}(v_j, v_k)$ between b_p and b.

Initial Migration Plan. In order to extract an initial plan (see Lines 6-7 of Algorithm 5) we find the shortest path in the time-graph (the dashed line in Figure 4). At each time step of the shortest-path a broker is determined that represents the expected host ($ep.H$) for an expected placement ep of the plan. Expected values for the bandwidths ($ep.B$) are taken from the bandwidth estimation for the operators outgoing stream. Furthermore, the starting time for the migration of immutable state ($ep.t_{im}$) is computed for each time t_i by subtracting the migration time t_{mig}. The first event of each queue $q \in Q$ ($ep.T_Q$) is then captured according to a window-model, e.g., [18]. Those windows determine the set of events in q that are required for the correlation at a discrete time, therefore if we model the windows position

```
 1: function createTimeGraph( $Time\text{-}Graph\ G$ )
 2:   $Targets \leftarrow getPossibleTargets()$
 3:   $s, f \leftarrow createStartAndStopNode(G)$
 4:   for $\forall ts \in TS$ do
 5:     for $\forall b \in Targets$ do
 6:       if shouldCreateVertex($b, G, ts + now$) then
 7:         $v \leftarrow createVertex(G, ts + now, b)$
 8:         for $\forall b_p \in Targets$ do
 9:           $t_{mig} \leftarrow estimateStateMigrationTime(b_p, b)$
10:           if checkEdgeCondition($b_p, b$) then
11:             $e \leftarrow edge(ts + now - t_{mig}, b_p, ts, b, G)$
12:             $w_e \leftarrow calculateEdgeWeight(e, t_{mig})$
13:           end if
14:         end for
15:       end if
16:     end for
17:   end for
18: end
```

Figure 6: Time Graph Maintenance

at t_i we can approximate the first required event from the windows bounds.

Coordination. An operator starts the coordination of plans by sending the plan to each neighbor (Line 10 of Algorithm 5). Then the operator waits until it received at least one feedback message from each neighbor until it starts the coordination with the selected targets (Line 24/26). A feedback can either be a changed plan of a neighbor or an acceptance of the plan. Note, that we even consider a plan that was sent concurrently by a neighbor and still in transmission when the plan coordination was started as feedback.

When a neighbor receives a plan, it re-evaluates its own plan before it decides on a feedback (Line 18). If the newly determined plan deviates from the current plan, the operator starts to coordinate its own plan, by sending a changed plan of its own as feedback (Line 10). Otherwise, the operator will send the feedback to all neighbors that recently updated their plans that it accepts their plans (Line 12/25). To reduce the number of coordination steps when neighbors concurrently generate plans, we ensure that a plan is only generated if all feedback is available and the plan is not coordinated with migration targets (Line 3-4). In the analysis we show that this coordination eventually terminates.

The coordination with migration targets happens in two steps. First, a request is sent to all migration targets, asking if i) they have enough resources available to host the operator ii) they can ensure the local latency restrictions. If this is not the case, the plan is rejected and re-evaluated at the initiator of the coordination, with a time-graph that does not create vertices for brokers that rejected the plan.

Although, these conditions are already checked when the time-graph is created, other plans of other operators compete for the resources. Therefore, plans have to reserve the resources at the migration target in a second step. This enables a broker to approximate its future resource utilization. If all future migration targets can execute the plan the resources are reserved and previous reservations of the previous plan are revoked.

Stability

Updated plans of neighbors bear the potential to make the plan of an operator unstable if the mobility is inaccurately captured or communication characteristics keep changing (e.g., diverging bandwidths). For example, if the operator generates a new plan with each location update of a source that instantly triggers a migration of an operator to a new host, we arbitrarily increase the network utilization.

To alleviate the mobility problem in the basic plan, we allow to restrict the number of time steps for the creation of the plan. The longer a consumer or source does not deviate from its predicted location, the more time steps can be considered for the time-graph. The intuition behind that can be seen in Figure 4. Edge weights that include migration costs are typically higher than for edges between the same broker. Therefore it is unlikely that an edge from any broker b_i to another broker b_j is considered for the shortest path if only few time steps are included, even if a potential migration target promises lower placement costs. At the same time we make sure that an operator eventually can migrate, even with a small number of time steps, by gradually decreasing the initial migration cost from the current host to neighbors the longer an operator stayed at a broker.

To ensure that a plan does not continuously change, only because of a small deviation in the measured bandwidth or latency, edge weights are only replaced in a time-graph if these measurements change beyond a threshold.

Properties

The number of brokers that are selected in Line 2 of Algorithm 6 has a severe impact on the performance of the live-migration system. Modelling all brokers, in a large-scale scenario with thousands of broker, is computationally costly, because the number of vertices and edges grows cubically. The number of vertices, per time step in the time-graph, grows linearly within each time step with the number of brokers n_b, thus the complexity is $O(n_b * |TS|)$. More severe is the number of edges, which grows quadratically per time step, because we connect each broker with all other brokers of the next time step. Thus the complexity is $O(n_b^2 * |TS|)$, which is cubic if the number of brokers n_b equals $|TS|$. Furthermore, each broker has to acquire information about the latency, bandwidth and available resources of these brokers, to determine the placement costs, which can only be determined by regularly sending messages between the broker.

Although selecting all brokers as migration targets gives a near-optimal solution to the mobile migration problem, we decrease the network and computation costs by only selecting the most relevant broker. To increase the scalability, we utilize the spatial-temporal locality property. Mobile objects move only within local bounds, e.g., do not connect to the system from London and seconds later from New York. Hence, good future migration targets are found close to the current broker that hosts the operator. Hence, each broker keeps coarse grained information on brokers responsible for far away locations, and fine-grained information on nearby broker.

4.2 Uncertainty-aware Migration Plans

So far, we presented plans as sequences of definite placements. However, since these sequences represent the predicted future, it comprises various uncertainties. Sources and consumers can change their movement pattern, e.g., vehicles changing their routes, or they hand-off between adjacent leaf broker at unexpected times. Data-rates of event streams can also vary, e.g., when an operator detects less events than expected.

Depending on how the system captures the movement of mobile devices, the actual future connection to a leaf broker is predicted more or less accurately. Simple dead reckoning mechanisms predict future locations based on the direction and speed of the last few reported locations, which is typically only accurate for nearby locations. The next sequence of leaf brokers and when a source or consumer connects to them is therefore highly uncertain. This can be improved by using more definite locations of navigation systems. Learning connection patterns between neighboring leaf broker gives a more accurate view than dead reckoning, but less accurate than the path of a navigation system. For example, while driving on a highway vehicles will connect with high probability to the same sequence of leaf broker that cover subsequent sections of that road and with a low probability take the next exit that may require to connect to another leaf broker.

Since data-rates of event streams can vary, it is possible that placing an operator ω at its planned migration target imposes higher costs compared to an optimal placement. In case the placement costs dominate over the migration costs, it is beneficial to stall the decision for a concrete migration target until it is certain that the placement can be improved. However, dependent operators, e.g., with large state, now require the information about all possible migration targets of ω to plan their migrations.

In this context we have to answer:

- What is a meaningful representation of an uncertain connection and migration decision?
- Given the uncertain representation of neighbors what are the best migration targets incorporated in a plan?

We address the question of the uncertainty-aware representation by expanding the plan to a *Markov chain*. The second one is addressed by a set of heuristics that aim to minimize the expected overall migration and placement costs of a single operator.

Uncertainty-aware Migration Plan Model

An uncertainty-aware plan is represented as *Markov chain*, as depicted in Figure 7. It allows for more than one expected placement at the same time step, connecting adjacent time steps with probabilistic state-transitions. We distinguish two kinds of uncertainty-aware plans. First, *temporally skewed migrations*, describe that instead of migrating at a fixed time step a mobile consumer/source or operator will initiate the migration within a temporal interval of discrete time steps. For example, a vehicle's connection change from b_1 to b_2 according to the plan in Figure 7 is performed with probability 0.5 at time t_1 and with probability 0.5 at t_2. Second, *spatially skewed migrations*, allow to describe multiple migration possibilities at the same time step, e.g., if a transition from b_1 to b_3 at t_2 in Figure 7 were included.

Creating the Uncertainty-aware Migration Plan

This section describes how the system creates uncertainty-aware plans for a consumer or source to indicate the possible future leaf broker and its data-rates. Furthermore, upon receiving uncertainty-aware plans, an operator has to adapt its own plan. To this end it can select from a pool of policies that aim to minimize the expected bandwidth-delay product but vary in their complexity.

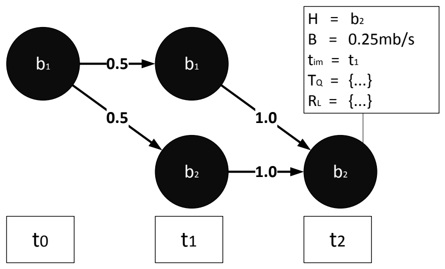


Figure 7: Uncertainty-aware Migration Plan

Planning for consumers and sources. Dead reckoning methods and navigation systems provide a sequence of time-stamped uncertain future locations[†] for mobile sources and consumers. This allows us to determine the expected placements *ep* of the plan for a set of future time steps $t_i \in TS$. Expected hosts ($ep.H$) are all future leaf brokers responsible for the future location at t_i. The average over the monitored recent traffic determines the expected bandwidth ($ep.B$). Probabilities for state transitions from previous time steps t_{i-1} are estimated based on the proportion of the overlap of the uncertain location with the leaf brokers area weighted by the extent of the uncertain location.

This method changes for learned connection patterns of consumers and sources. These learned patterns describe how probable it is for a source or consumer to change its connection to another leaf broker after it stayed connected for a certain amount of time to its current leaf broker. Such learned probabilities are maintained in a graph where each vertex represents a broker and has an edge to all neighboring brokers the source or consumer can connect to next. Each edge in the graph is annotated with the probability to change the connection and the average time a source or consumer is connected to a broker before it connects to the neighbor.

Such a graph can then be exploited to find a sequence of expected hosts $ep.H$ for an uncertainty-aware plan. A simple algorithm follows paths in the graph sequentially from broker to broker, starting from the current leaf broker of a mobile source or consumer. The algorithm keeps track of the annotated times of edges while traversing the graph and reports a broker as $ep.H$ at a $t_i \in TS$ if the next edge traversal increases the sum of annotated times on that path beyond t_i. State-transition probabilities of the uncertainty-aware plan are equivalent to the probability that a connection changes. Thresholds on the probability that a path represents the actual connection pattern can reduce the number of states.

Policies of operators. In the following section we present policies to process and create uncertainty-aware plans.

Naive Policy: Since each sequence of state transitions in a neighbors probabilistic plan resembles the sequence of brokers of the basic plan creation, the naive approach is to create a plan using multiple time-graphs; in particular, one time-graph for each possible combination of these individual sequences. For example, if the operator ω_D of our running example receives the plan of Figure 7 from the source and a basic plan from ω_F it creates one time-graph with the sequence $seq_1 = \{b_1,b_2,b_2\}$ and one with the sequence $seq_2 = \{b_1,b_1,b_2\}$ of ω_Ds plan. All shortest paths in these

[†]Note, this can be a typical GPS normal distribution

time-graphs are then combined to a plan, where each shortest path represents one sequence of state transitions in the resulting plan. Consider the case when the resulting shortest path of the time-graph for seq_1 leads to the sequence $seq_3 = \{b_3,b_4,b_4\}$ and for seq_2 to $seq_4 = \{b_3,b_3,b_4\}$. Since the probability for ω_F to follow the basic plan is 1 and the probability of the source to connect according to the sequence seq_1, respectively seq_2, is 0.5, each of those resulting sequences has the probability of $1 * 0.5$ to represent the actual migration sequence of ω_D. When both are combined to a single plan it looks structurally similar to Figure 7. This policy allows an operator to accurately follow the uncertain movement of a neighbor by creating an uncertain plan for the operator. However, it yields a high computational cost, because it requires to maintain many time-graphs.

Weighted Sum: This policy modifies the cost function to determine the placement costs C_{pla} in a time-graph, in order to create a plan that minimizes the overall expected placement and migration costs. It uses the weighted sum over the bandwidth-delay products to all expected hosts at a time $t_i \in TS$ in a neighbor's plan to determine placement costs $C_{pla}(v_j)$ for a vertex v_j at that time. The weight is the individual probability that an expected placement is chosen, i.e., the summed product of all state transitions on sequences to that expected placement, e.g., 0.5 for both expected placements at t_2 and 1 for the expect placement at t_3 in Figure 7. The computational costs are far lower than for the naive approach.

Temporal Adjustment: Since temporally skewed migrations of neighbors only indicate that the migration between any pair of brokers b_j, b_k can vary in time, the shortest path is determined in a time-graph that only considers one sequences of transitions between those in the neighbors plans. For example only for sequence $seq_1 = \{b_1,b_2,b_2\}$ of the example in Figure 7. An operator can then calculate a temporally skewed plan of its own or use this plan to find the single sequence of brokers with the lowest expected bandwidth-delay product. This is done by determining the probability that the operator is allowed to migrate at another time step in a predefined temporal interval $[t_i - t_s, t_i + t_e]$, instead of the time t_i at which the ith migration occurs according to the shortest path. Let $seq_3 = \{b_3,b_4,b_4\}$ be the initial resulting sequence, then for a t_s of 1 $seq_3' = \{b_3,b_3,b_4\}$ and $seq_3'' = \{b_4,b_4,b_4\}$ are also considered. The operator can then choose between selecting the sequence with the lowest expected overall bandwidth-delay product, according to the weighted sums at each time step, as plan or to use all of them as a temporally skewed plan. Like the naive approach this approach also allows an operator to follow the uncertain movement of a neighbor, however, with a smaller accuracy and computational cost.

Changes in data-rates: The last policy is the only one that creates an uncertainty-aware plan without previously receiving such plans from neighbors. Since a time-graph comprises estimated costs on future placements and migrations, the actual costs at the time of an estimated migration might differ. Recall that an operator with small state can defer its migration to such a time. In order to reflect these scenarios in the plan, we select the k shortest paths from a single time-graph, since they are the most likely paths to actually have the lowest costs. For example, due to a deviation in the bandwidth it can be good to also consider the edge from b_1 to b_2 at t_1 in Figure 4.

```
 1: upon timeout migrationTimer($time\ t$)
 2:   $nextMigration \leftarrow$ findNextTarget(t)
 3:   if $nextMigration.t_{im} = t \land \neg stateMigrated$ then
 4:       trigger send immutalbeState to $nextMigration.H$
 5:   end if
 6:   for $\forall tq \in nextMigration.T_Q$ do
 7:       if shouldMigrate($tq$) then
 8:           initiateMutableStateTransfer($tq$)
 9:           initiateStreaming($tq$)
10:       end if
11:   end for
12:   if allStateAtTarget($nextMig.t_i$) then
13:       stopOperator()
14:       trigger send start() to $nextMigration.H$
15:   end if
16: end
```

Figure 8: Execution of a Migration Plan

Each of these k paths represents a sequence of transitions in the plan. The probabilities that these paths represent the actual shortest path are determined by estimating the probability that the estimated cost for one of these paths is lower than for all the other paths and therefore describes the actual migration behavior. We therefore understand the sum of all weights W_{tg} on these paths as random variable X_{tg}. The confidence interval $(min(X_{tg}), max(X_{tg}))$ is given by the interval that determines how far bandwidths and latencies can deviate before the time-graph is created anew. However, without the knowledge of the real distribution within that interval, we can either assume that values are distributed according to a rather typical distribution, such as poison or uniform, learn those distributions, or use hints from the application on the actual distribution of bandwidths from in and outgoing streams. The continuous solution for the probability, that the j-th path has the lowest cost is then:

$$P_{min}(X_j) = \int_{y=min(X_j)}^{max(X_j)} \prod_{x \neq j} P(X_x > y)$$

For example, the weights for all depicted paths in Figure 4 including only brokers of b_1 and b_2 accumulate to the same costs. Therefore the probability for both paths in the interval $[t_1, t_2]$ is 0.5.

4.3 Executing a Migration Plan

Algorithm 8 allows the operator to implement the migration according to the probabilistic state transitions in a plan. It decides on one of the possible placements from the plan and informs the live-migration system about when to transfer and initialize an operator at the migration target. The algorithm serializes the migration of immutable and mutable state, while it continues processing at the previous broker until the migration target starts processing.

At each time step $t \in TS$ (see Line 1) of the plans, the algorithm checks if any possible migration needs to be initiated. It starts at the state that models the current placement, e.g., b_1 at t_1 in Figure 7. For the next possible migration according to this plan, e.g., from b_1 to b_2 at t_2, the operator checks if it has to start transferring the immutable state before the next time step. For uncertainty-aware plans this is checked for the sequence that currently has the lowest expected bandwidth-delay product (see Line 2-3). The migration itself starts out by sending the VM and immutable state to the selected migration target (see Line 4). This is skipped if the immutable state already started to be transferred in a previous time step.

In the next step (see Line 6-11), we determine if parts of the mutable state have to be transferred before the next step. This is done by estimating for each queue Q the size of the mutable state that is currently available in all queues starting from T_Q on and how long it takes to migrate these states. In this step, we also inform the neighbors in the operator-graph that they have to start streaming to the migration target and eventually stop streaming to the current host. They in return inform the migrating operator about the first event they transferred to the migration target, which indicates when to stop the streaming of mutable state from the current host to the migration target.

In a final step (see Line 12-15), the previous broker stops the processing at the time when mutable and immutable state is available at the migration target. Finally, it initializes the operator, in a rollback-recovery like fashion [17] at the target.

5. ANALYSIS

5.1 Locality of Plan Generation

The overhead in terms of messages to generate a plan depends on where the plan is generated. Which is either locally at each operator ω or at a central coordinator, e.g., at a dedicated node in the cloud. Note that a time-graph with vertices for all combinations of placements of operators increases the total number of vertices to the power-set. This is why it is more scaleable, even for a central coordinator, with a global view on all operators, to maintain a single time-graph for each operator.

A locally generated plan imposes the overhead to exchange plans (and feedback) with neighbors, which happens in average at a frequency $f_p(\omega)$ with an average size m_p for each message. In the central case all changes in measured and estimated bandwidths have to be sent to the coordinator, which happens in average at a frequency of $f_m(\omega)$ and with an average size of m_m for each message. Stable plans are sent to individual operators with a frequency $f_s(\omega)$. Other messages, i.e., to reserve resources at targets and collect information of latencies on links between neighbors, are nearly the same for both possibilities. A locally generated plan pays off, iff:

$$\sum_{i=0}^{|\Omega|} f_p(\omega_i) * m_p > \sum_{i=0}^{|\Omega|} f_m(\omega_i) * m_m + f_s(\omega_i) * m_p$$

5.2 Threshold for migration generation

In this section we derive a threshold, that determines when a potential second path in the time-graph is shorter than the selected shortest path. This gives a bound on when it is safe to not generate a new plan. Let $(v_1, ..., v_v)$ be the currently selected minimal path and $(v'_1, ..., v'_v)$ any other path. Let D be difference in their overall costs of the edge weights w.

$$D = \sum_{i=0}^{|E|-1} w((v_{i+1}, v_i)) - \sum_{i=0}^{|E|-1} w((v'_{i+1}, v'_i))$$

Only if D is positive, a second path can be shorter. Thus if the change in all weights stays in the bounds of the following threshold at the *ith* edge, D is guaranteed to be negative and the selected path is shorter.

$$\frac{|w((v_{i+1}, v_i)) - w((v'_{i+1}, v'_i))|}{2}$$

5.3 Termination

An important property of the coordinated plan generation algorithm is that it eventually terminates, i.e., no new shorter path is selected, iff the estimated latencies on links and bandwidths incorporated in a time-graph do not change while the plan is generated. We claim that, with each newly selected shortest path for the plan generation of an operator the overall costs C_{tot}, the sum over all placement and migration costs, strictly monotonically decreases. We show the termination property by a proof of contradiction.

Proof. Assume that the total costs C_{tot} increase to C'_{tot} after a new shortest path is selected for an operator ω. The only links that can now increase the placement costs are those directly involved in sharing event streams with neighboring operators of ω. The only migration costs that can be increased are those imposed by ω. Since these costs are used to calculate the edge weights of the time-graph, at least the previous shortest path would have been shorter. □

6. EVALUATION

We implemented the migration approach with the Omnetpp simulator [26]. The traffic simulation package SUMO [1] enabled us to model realistic traffic patterns of vehicles on an OpenStreetMap graph [11]. The approach was tested with a generic operator graph resembling Figure 1(a), that allowed us to extract meaningful thresholds[‡].

Brokers were organized in a hierarchical tree data-structure, where each tier contained a dynamic number of $nb_x * nb_y$ simulated cloud or fog-nodes. With each level in the hierarchy the computational power increased quadratically, while the latency between neighbors in the hierarchy were similar. Vehicles were always connected to a leaf-node of the tree that managed the area where the car was located in. Over the course of 1000 simulated seconds approx. 1000 vehicles drove in an area of the size $7.7x3.5km$. Each connected vehicle published approx. one event per second. Each measurement was taken approx. 5 times.

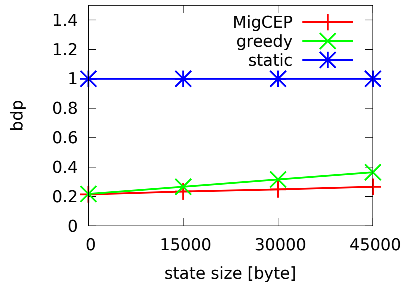
We initially distributed all virtual brokers in the broker hierarchy randomly and tested three possible migration approaches: i) the *static* approach that didn't perform any migration, ii) the *greedy* approach that greedily selected every few seconds the broker with the best placement cost, and iii) our *MigCEP* approach.

6.1 Impact of state and streaming size

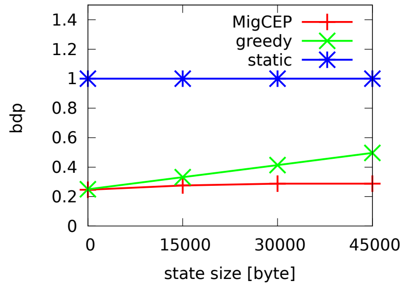
Since the *MCEP* approach is designed to consider the state and streaming size for an optimal sequence of migrations, we studied their influence on the main performance criteria, the network utilization. Furthermore, an increase in the number of brokers on the same area increases the number of hand-overs in the system, the main source of dynamics. Therefore, we also evaluated the impact of the number of brokers in the hierarchy on the network utilization.

In the experiment, we gradually increased the *event size* of one of the sources in the operator graph from 50 to 250 bytes

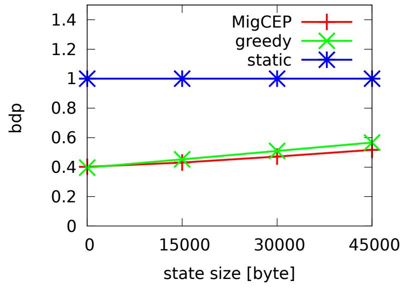
[‡]For simplicity we omitted source c. We parametrized the operator graph regarding the state size, event size, and detection rate.



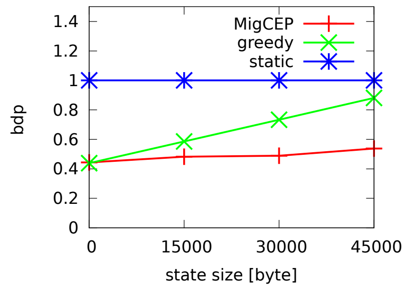
(a) Event Size 50; 4*4 broker/level



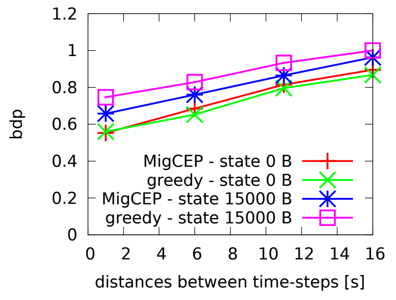
(b) Event Size 250; 4*4 broker/level



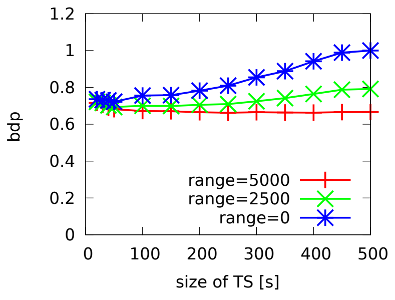
(c) Event Size 50; 10*4 broker/level



(d) Event Size 250; 10*4 broker/level



(e) Distance between TS



(f) Varying ranges and |TS|

Figure 9: Evaluation of basic time-graph based approach

and the immutable *state size* of the operator that received those events from 0 bytes to 60000 bytes. The number of brokers varied from 21 fog-nodes in a quad-tree with a single cloud data-center root to a tree with one simulated cloud data-center as root and a flat topology of 40 fog-nodes. The information from the simulated navigation system of the vehicle was used to generate plans for vehicles. On the y-axis in Figure 9 (a-d) the results of the average bandwidth-delay product (bdp) is depicted, relative to the *static* approach. The x-axis depicts the immutable state size in bytes.

The results demonstrate the strength of our approach in both broker hierarchies. Our approach outperforms by far the static approach, since it adjusts the placements. While the resulting migration and placement costs are comparable to the greedy approach when the size of immutable state is low, the benefit increases the larger the immutable state is, since our approach amortizes the migration costs. The system had to adapt the placement of the operator more often in the case of larger event sizes (Figure 9(b) and 9(d)) which is why the benefit is more distinguished in those cases. The overhead averaged on low 27 additional coordination messages (resource reservations, plan updates, and feedback) per coordination — independent of event and state size.

6.2 Impact of vertices in the time-graph

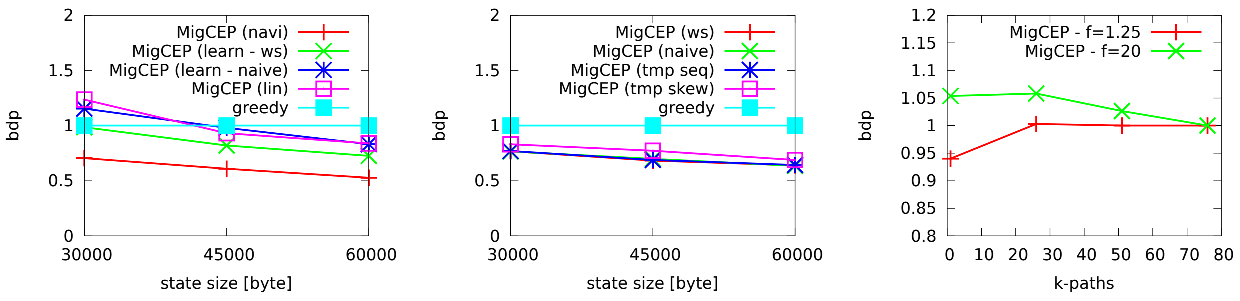
The performance of the plan generation depends on the number of vertices in the time-graph. This number changes with the granularity of migration times in a sequence of time-steps TS and the number of selected brokers.

The distance between time-steps dictates the system on when it can perform a migration. The longer the distance, the less vertices in the time-graph; however, the less chances to find the optimal time for a migration. Figure 9(e) depicts the results, relative to the maximal measured *bdp*, for varying distances between 1 and 16 seconds, effectively reducing the number of vertices by $\frac{1}{4}$ with each larger step. It demonstrates that the increase in the *bdp* is linear, while the number of vertices multiplicativly decrease.

Moreover, the number of time-steps of TS dictates how far into the future a prediction is made. In the experiment we increased this size from 10 to 500. We also restricted the number of brokers that are considered for the time-graphas possible target, by only selecting nearby brokers that manage ranges that are no more than 0, 2500, or 5000 meters away. This includes brokers in higher or lower levels of the hierarchy responsible for the same range. Figure 9(f) shows that a larger range gives a better prediction since more brokers were involved. However, the performance only gracefully degrades with fewer brokers. Few time-steps limited the opportunity for planning migrations and too many time-steps reduce the chance to place the operator on the right broker in the future.

6.3 Impact of uncertainties

The future placements typically encounter a lot of uncertainties, depending on how the mobility pattern and communication characteristics are captured. The general setup to test the policies dealing with these uncertainties (Section 4.2) was to deploy both ω_Fs as state-less operators and ω_D as state-full operator. For the *naive* and *weighted sum* policy (ws), we tested the three methods to capture mobility patterns, uncertain locations from the *dead reckoning* approach (linear), certain locations that could stem from a *navigation* system (navi), and *learned* transitions between leaf broker (learned). For the *temporal* policies we restricted the simulation to the *learned* temporal transition on the path of the navigation system, since the policy is only expressive for this capture method. To test the k-shortest path approach we also randomly increased and decreased the event-sizes. Hence, the placement of one of the ω_Fs had to be constantly adapted for an optimal placement.



(a) Spatially Skewed Plans (b) Temporally Skewed Plans (c) Changing data-rates

Figure 10: Impact of uncertainty

The x-axis of Figure 10(a) and Figure 10(b) depicts the varying state size. The average bandwidth-delay product depicted on the y-axis shows the effect of different methods to capture the mobility using different policies on the network utilization; using the greedy approach as baseline. Each policy increased the network utilization less severe than the greedy approach after the state size crossed a specific threshold. That threshold is the point where the reduction in the migration rate makes up for less optimal anticipated placements. Depicted in Figure 10(c) are the results for different numbers of sequences selected form the k-shortest path approach, where the frequency f in the change in the event-size ranged from every 20 seconds to every 1.25 second. The results are depicted relative to the *bdp* value of the largest k for each frequency. For the high-change rate, more paths increased the possibility for wrong placement decisions, however, for slow change rates the system had a better chance to adapt itself.

7. RELATED WORK

The placement of operators has already been studied in context of DCEP [7, 20, 22, 23]. However the main focus of these work are systems where sensors and consumers aren't mobile and the communication characteristics rarely change. Therefore none of the systems considered the impact of migration costs on the placement, nor do they have to consider the mobility of a consumer, which influences the time when it is required to trigger a migration.

Previous work in DCEP focused on uncertainty in detecting events, e.g., on noisy events captured by sensors [16] or predicted future events [10]. However, none of them studied the impact on the system resources when planing future placements with uncertain location information.

Mobility-aware publish-subscribe systems [14,15] dynamically adapt filter operators to new access points of publishers or subscribers. However, filter operators are stateless and therefore none of these systems considers the migration cost itself in the optimization.

Live migration [12,19] in cloud computing environments is used to increase data-access locality, energy saving, or load balancing. A typical goal is to reduce the downtime, the time during which the execution is stopped. On the one hand, pre-copying techniques [4] copy disk-images, i.e., the complete state of the operator, and memory-pages in advance before the processor state. This can arbitrarily increase the bandwidth, because events that are potentially deleted from the queues Q as soon as the VM starts at the target are also migrated. On the other hand, post-copying [12] techniques transfer first the processor state and then the pages. This can increase the latency if the next required page is not migrated yet, even to such an extent, that latency restrictions are not preserved.

Content-delivery networks improve end-user performance, data availability, and reduce server load by migrating data. They have been considered for mobile environments [2] and cloud environments [27]. However, they do not have to consider dependencies between different operators.

8. CONCLUSION

In this paper we presented methods to improve the overall network utilization for real-time applications with defined end-to-end latency restrictions in future mobile infrastructures. Using a MCEP system as example, we demonstrated that it pays off to plan migrations ahead of time according to the mobility of users or dependent migrations. Costly migrations of state, in terms of network utilization, can be amortized by selecting suitable targets in a time-graph data structure that models the expected costs. Furthermore, we presented how application knowledge improves live-migration systems. Execution environments that are typically used in DCEP provide the possibility to find a better serialization of operators, because they allow a live-migration system to infer which state has to be transferred.

Our ongoing work will focus on further optimization possibilities. We tested the system with only simple prediction mechanisms for data-rates, however, more sophisticated ones could greatly improve the performance. So far, we did not consider the fact that different operators might require the same mutable state—an overlapping set of ingoing event-streams. Finding such overlaps would allow the system to coordinate the migration of multiple operators while migrating the immutable state only once, thus further improving the network utilization.

Acknowledgment

This work is supported by contract research "CEP in the Large" of the Baden-Württemberg Stiftung. The authors would like to thank the reviewers and D. Lillethun for their helpful comments.

9. REFERENCES

[1] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz. SUMO - Simulation of Urban MObility: An Overview. In *Proc. of 3rd Int. Conference on Advances in System Simulation (SIMUL)*, pages 63–68, Barcelona, Spain, Oct. 2011.

[2] M. M. Bin Tariq, R. Jain, and T. Kawahara. Mobility Aware Server Selection for Mobile Streaming Multimedia Content Distribution Networks. In F. Douglis and B. D. Davison, editors, *Web Content Caching and Distribution*, pages 1–18. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[3] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog Computing and Its Role in the Internet of Things. In *Proc. of 1st MCC workshop on Mobile Cloud Computing*, pages 13–16. ACM, 2012.

[4] R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schiöberg. Live Wide-Area Migration of Virtual Machines Including Local Persistent State. In *Proc. of 3rd Int. Conference on Virtual Execution Environments (VEE)*, pages 169–179. ACM, 2007.

[5] A. T. Campbell, S. B. Eisenman, N. D. Lane, E. Miluzzo, R. A. Peterson, H. Lu, X. Zheng, M. Musolesi, K. Fodor, and G.-S. Ahn. The Rise of People-Centric Sensing. *IEEE Internet Computing*, 12(4):12–21, 2008.

[6] S. Chakravarthy and D. Mishra. Snoop: An Expressive Event Specification Language For Active Databases. *Data & Knowledge Engineering*, 14(1):1–26, 1994.

[7] G. Cugola and A. Margara. Deployment Strategies for Distributed Complex Event Processing. *Springer Computing*, 95(2):129–156, 2013.

[8] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A Decentralized Network Coordinate System. In *Proc. of 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 15–26. ACM, 2004.

[9] C. du Mouza, W. Litwin, and P. Rigaux. SD-Rtree: A Scalable Distributed Rtree. In *Proc. of the 23rd Int. Conf. on Data Engineering (ICDE)*, pages 296–305. IEEE, Apr. 2007.

[10] Y. Engel, O. Etzion, and Z. Feldman. A Basic Model for Proactive Event-driven Computing. In *Proc. of 6th ACM Int. Conference on Distributed Event-Based Systems (DEBS)*, pages 107–118. ACM, 2012.

[11] M. Haklay and P. Weber. OpenStreetMap: User-Generated Street Maps. *IEEE Pervasive Computing*, 7(4):12–18, Dec. 2008.

[12] M. R. Hines, U. Deshpande, and K. Gopalan. Post-Copy Live Migration of Virtual Machines. *SIGOPS Oper. Syst. Rev.*, 43(3):14–26, July 2009.

[13] K. Hong, S. Smaldoney, J. Shin, D. Lillethun, L. Iftodey, and U. Ramachandran. Target Container: A Target-Centric Parallel Programming Abstraction for Video-based Surveillance. In *Proc. of 5th ACM/IEEE Int. Conf. on Distributed Smart Cameras (ICDSC)*, pages 1–8, Aug. 2011.

[14] S. Hu, V. Muthusamy, G. Li, and H.-A. Jacobsen. Transactional Mobility in Distributed Content-Based Publish/Subscribe Systems. In *Proc. of 29th IEEE Int. Conf. on Distributed Computing Systems (ICDCS)*, pages 101–110, June 2009.

[15] K. Jayaram, C. Jayalath, and P. Eugster. Parametric Subscriptions for Content-Based Publish/Subscribe Networks. In *Proc. IFIP/ACM/USENIX Conf. on Middleware*, pages 128–147, 2010.

[16] G. G. Koch, B. Koldehofe, and K. Rothermel. Higher Confidence in Event Correlation Using Uncertainty Restrictions. In *Proc. of 28th Int. Conf. on Distributed Computing Systems (ICDCS) Workshops*, pages 417 –422, June 2008.

[17] B. Koldehofe, R. Mayer, U. Ramachandran, K. Rothermel, and M. Völz. Rollback-Recovery without Checkpoints in Distributed Event Processing Systems. In *Proc. of 7th ACM Int. Conf. on Distributed Event-Based Systems (DEBS)*. ACM, 2013.

[18] B. Koldehofe, B. Ottenwälder, K. Rothermel, and U. Ramachandran. Moving Range Queries in Distributed Complex Event Processing. In *Proc. of 6th ACM Int. Conf. on Distributed Event-Based Systems (DEBS)*, pages 201–212. ACM, 2012.

[19] R. K. K. Ma and C.-L. Wang. Lightweight Application-Level Task Migration for Mobile Cloud Computing. In *Proc. of 26th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, AINA '12, pages 550–557. IEEE Computer Society, 2012.

[20] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer. Network-Aware Operator Placement for Stream-Processing Systems. In *Proc. of 22nd Int. Conf. on Data Engineering (ICDE)*, pages 49–60. IEEE Computer Society, 2006.

[21] P. Pietzuch, B. Shand, and J. Bacon. Composite Event Detection as a Generic Middleware Extension. *IEEE Network*, 18(1):44–55, Feb. 2004.

[22] S. Rizou, F. Dürr, and K. Rothermel. Solving the Multi-operator Placement Problem in Large-Scale Operator Networks. In *Proc. of 19th Int. Conf. on Computer Communication Networks (ICCCN)*, pages 1–6. IEEE Communications Society, Aug. 2010.

[23] B. Schilling, B. Koldehofe, and K. Rothermel. Efficient and Distributed Rule Placement in Heavy Constraint-Driven Event Systems. In *Proc. of 13th IEEE Int. Conf. on High Performance Computing and Communications (HPCC)*, pages 355 –364, Sept. 2011.

[24] Z. Sebepou and K. Magoutis. CEC: Continuous Eventual Checkpointing for Data Stream Processing Operators. In *41st Int. Conf. on Dependable Systems Networks (DSN)*, pages 145–156, June 2011.

[25] J. Shin, R. Kumar, D. Mohapatra, U. Ramachandran, and M. Ammar. ASAP: A Camera Sensor Network for Situation Awareness. In *Proc. of 11th Int. Conf. on Principles of Distributed Systems (OPODIS)*, pages 31–47. Springer-Verlag, 2007.

[26] A. Varga. The OMNeT++ Discrete Event Simulation System. In *Proc. of the European Simulation Multiconference (ESM)*, June 2001.

[27] P. Wendell, J. W. Jiang, M. J. Freedman, and J. Rexford. DONAR: Decentralized Server Selection for Cloud Services. In *Proc. of ACM SIGCOMM*, SIGCOMM '10, pages 231–242, New York, NY, USA, 2010. ACM.

[28] O. Wolfson, A. P. Sistla, S. Chamberlain, and Y. Yesha. Updating and Querying Databases that Track Mobile Units. *Distributed and Parallel Databases*, 7:257–387, July 1999.