

Institut für Parallele und Verteilte Systeme
Universität Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Studienarbeit Nr. 2359

Complex Event Processing in the Cloud

Eva Hoos

Studiengang: Informatik

Prüfer: Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel

Betreuer: Dipl.-Inf. Björn Schilling

begonnen am: 15. November 2011

beendet am: 16. Mai 2012

CR-Klassifikation: C.2.4

Contents

1	Introduction	7
1.1	Task description	9
2	Foundation	11
2.1	Event Processing	11
2.2	Complex Event Processing	12
2.2.1	Esper	13
2.3	Cloud Computing	16
2.3.1	Infrastructure-As-A-Service	18
3	Related Work	21
4	The System Model	25
4.1	The CEP Network	25
4.1.1	Eventrate	25
4.1.2	Virtual Node	27
4.1.3	Event Stream Splitting	27
4.1.4	Rule Cluster	27
4.2	Cloud Provider	28
5	Resource Manager	29
5.1	Decision Criterion: Critical Workload	30
6	Evaluation	33
6.1	Implementation	33
6.2	Error description	34
6.3	Test Series	35
6.3.1	Test Series 1: Different Gradients	35
6.3.2	Test Series 2: Different Thresholds	38
6.3.3	Test 3: Long-term observation	38
6.3.4	Conclusion	40

7 Summary and Future Work	43
Bibliography	45

List of Figures

1.1	Rule Network	8
1.2	CEP Network	9
2.1	Event-based system [MFP]	12
2.2	Esper Process Model [Espo8]	14
2.3	The Cloud Computing Stack [VBB11]	17
2.4	The lifecycle of a virtual machine [ER11]	19
3.1	Aurora system model [MHM ⁺ 03]	21
3.2	Kleiminger's approach [KKP11]	22
3.3	Hummer's approach [HLSD11]	23
4.1	Old mapping	25
4.2	New mapping	26
4.3	The-function-rate()	26
4.4	Cluster	28
5.1	Allocation process	32
5.2	Deallocation process	32
6.1	Error Area	35
6.2	Test b: Increasing 100 Events per second	37
6.3	Increasing 30 Events per second	37
6.4	Increasing 200 per second	39
6.5	Test a: Threshold = 0,9	39
6.6	Test b:Threshold = 0,7	41
6.7	Long-term observation	41

Chapter 1

Introduction

Sensors measure characteristics of the environment as temperature or distance. They produce data and send it to a service that manages this data. Many applications use sensors to get information about the environment. Some of them like GPS and accelerometer are built in Smart Phones and Tablet PCs to determine the location and position of the device. Others are used to automate processes. For example, with Cameras and RFID (radio-frequency identification) chips it is possible to track a product through a manufacturing process.

Sensors are used to recognize events as earthquake or position change. In general, events are anything that happens. Often the term event is also used as synonym for event message. An event message is the representation of an event for the purpose of computer processing [LS08]. Events are processed on the basis of rules like deleting, transforming and aggregating. In this way complex events are generated with a higher semantic level. Often not a simple event is important, rather than the aggregation of events. For example, for the start of an airplane it is important to know that all baggage is on board, not a particular bag. Processing events to complex events as well as computing on complex events is called Complex Event Processing (CEP).

A possible application for CEP is a baggage management system at the airport. The baggage management system is responsible for the correct routing of the baggage. Therefore, baggage is tagged at the Check-In with RFID (radio-frequency identification) tags. When a bag passes a tag reader device, a message containing the position and time is reported to route the bag to its destination. Thus, the tag reader recognize the event "baggage passes".

Applications in CEP are realized by a rule network as shown in Figure 1.1. The inputs are simple events and they are processed to complex events. With each rule the number of the events decreases but the expressiveness increases.

Events do not occur independent from each other, because events are related to other events by time, causality, and aggregation. Time dependency is based on timestamps, which make it possible to order events. If the timestamp of event A is greater than of event B, then event A happens before B. For example, if bag A reaches the reader before bag B, then A and B have a time related order. On the other hand causal dependency means that event A had to

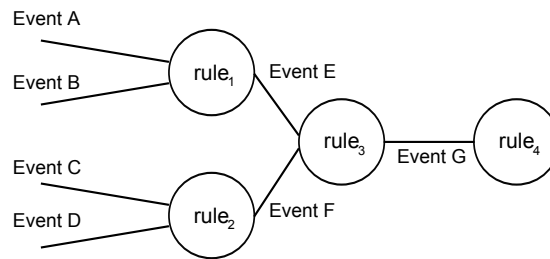


Figure 1.1: Rule Network

happen in order that B happen. For example, only after the plane arrives, the baggage on board can be tagged by a device at the airport. The third dependency aggregation applies to complex events, event A is created by using events B_i . If all baggage is on the plane, then the event “all baggage on board” can be reported and the pilot can start.

According to the complexity of the rule and the number of events been processed the workload varies. The higher the complexity of the rule the higher the workload is. Furthermore, the more events have to process, the higher is the workload. It follows, during the processing, the workload is not constant, there are times when more events should be processed and times when less events should be processed. A reason for such peaks is, that events are related to each other and these events often occur intermediately. For instance, if a plane arrives, lots of new baggage has to be routed. Therefore, more and more bags pass tag reader devices and new events arise. These events should be processed in real time to avoid that bags get lost or routed wrongly. Hence, within shortest possible time more compute power is needed to process all events. If the events could not be processed in real time, either the events are stored and processed with delay or they have to drop which can leads to wrong results. On the other hand there are times when fewer events have to process. For example, at many airports there is a night flight ban. Thus, no baggage has to be routed at this time.

It follows, that a scalable system is required, where it is possible to allocate new resources if more computing power is required and to scale down, if less computing power is needed.

In a centralized system all events are processed by a single computer but this is not scalable, because the resources are limited. If the workload increases, there is no possibility to handle the additional workload. Therefore, nowadays applications are distributed as shown in Figure 1.2, so that the workload can be distributed among the network participants. The rules are distributed to several physical nodes. Each node runs a CEP engine, which performs multiple rules. However, in standard distributed systems it takes too long to allocate new resources in order to handle rapidly increasing workload.

A solution for flexible resource allocation is Cloud Computing. From a Cloud provider, consumers can acquire virtual computing resources on demand. Capabilities are available over the network. More precisely, the provider’s computing resources are pooled to serve multiple consumers’ requests using a multi-tenant model with different physical and virtual

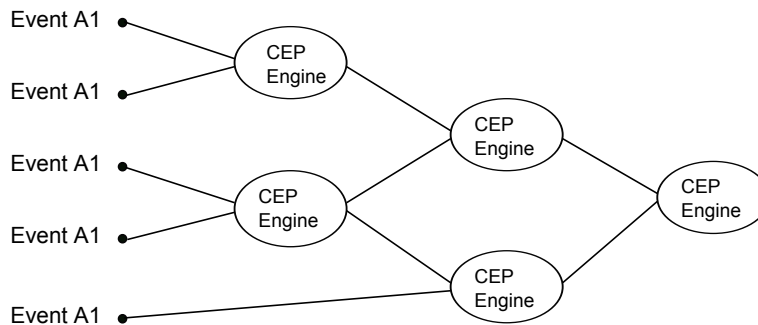


Figure 1.2: CEP Network

resources. The resources can be dynamically assigned and reassigned according to consumer demands [VBB11]. The advantage is that the provision is rapid, flexible and the consumer pays only for the capacity of the provisioned resource at a particular time. A consumer can acquire as many virtual machines as he needs. Hence, virtualization is an important topic in Cloud Computing. A virtual machine acts like a real one including the components, storage, processing power, memory and network.

Cloud Computing can be used to handle workload peaks of a CEP application. If there are lots of events to be processed, then the computing power of the virtual machine has to be increased. If the physical limit is reached, then a new virtual machine has to be allocated. On the other hand it is possible, that the resource can be deallocated to reduce costs.

1.1 Task description

In this work a new approach is developed to handle peaks in the workload by using a Cloud environment. The Cloud environment makes it possible to allocate new resources quickly in order to process more events. Storing events in a queue should be avoided, because to less resources are available. If the system recognizes that more resources should be allocated. It should also be consider, that no unused resources is allocated, because in the Cloud resources have to be paid as long as the resources are allocated.

At first, a system model will be developed that makes it possible to deploy the CEP network in the Cloud environment. Furthermore, the system will be able to reconfigure itself in order to add new resources as well as to delete existing resources and to balance the workload on all existing resources.

In the second step, an algorithm will be developed, which decides if new resources have to be allocated and when resources have to be deallocated. The algorithm will ensure, that the new resource is available when it is needed, which means without the new resource, the events will be stored in the queue. Thus, that as less as possible resources are allocated, which means a resource can be deallocated, if it has no effect on the event processing.

1 Introduction

The last step will be an evaluation of the methods in different situations to find out what the possible error sources are, in what situations can this solution be used and what are the advantages and disadvantages. Furthermore, it should also be considered, which possibilities exist to improve this approach.

Foundation

2.1 Event Processing

This chapter covers the foundation of the term event processing which includes all procedures executing on events. At first, it is essential to know what an event is. A short explanation is already given in the introduction as well as some examples. There are lots of different definitions in the field of event processing because it was developed in various areas including business activity monitoring, sensor networks and market data [EBo9]. The first approach for standardization is a glossary [LS08] published by the Event Processing Technology Society (EPTS). They define an event as follows:

Definition 1. (EVENT): An event is anything that happens, or is contemplated as happening.

This definition included that an event can be real, for example an earthquake or a landing airplane, or artificial as changing a state in a database or in a finite state machine. In order to process an event it is mandatory to have a representation that can be handled by a computer. The EPTS defined this as follow:

Definition 2. (EVENT OBJECT, EVENT MESSAGE, EVENT TUPLE): An object that represents, encodes, or records an event, generally for the purpose of computer processing.

There may be several event messages belonging to the same event, because several devices recorded the same event. In general event processing is defined as follows:

Definition 3. (EVENT PROCESSING): Computing that performs operations on events, including reading, creating, transforming and deleting events.

Processing events is often done in event-based systems shown in Figure 2.1. There are three main components: producer, consumer and notification service. A producer recognizes events and passes the notification to the notification service. In the context of this work, a notification is an event message. The consumer registers at the notification service for

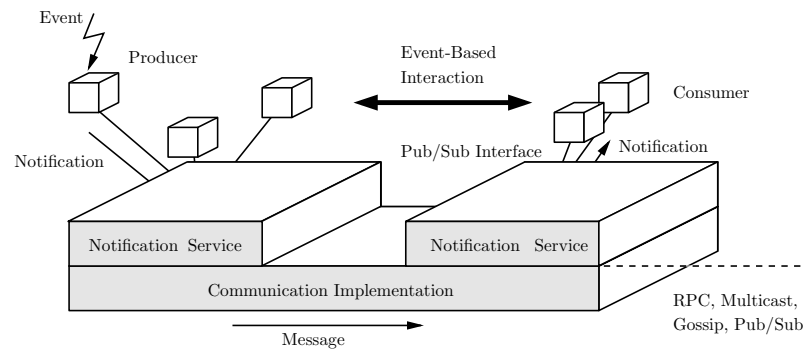


Figure 2.1: Event-based system [MFP]

particular events and receives them from the notification service, which acts as a mediator between the components and transmits the notification from the producer to all consumers that have notified for it. The consumer does not have to know the producer of the event as well as the consumer is unknown to the producer. This means consumer and producer are decoupled.

For example, in a baggage management system the RFID readers are producers. They detect new baggage, record the events and send the information to the baggage management by a notification service. The event messages are processed and the result can be passed as a new event message, for example to give the signal “all baggage on board” .

2.2 Complex Event Processing

Event processing on simple events have some problems, because there are a lots of simple events with low semantic expressiveness. To get the desired result, lots of events have to be considered. Hence, it is difficult to recognize complex correlations and lots of computing power is required to process all events. Therefore, Complex Event Processing was developed. Complex events have a higher semantic level and they aggregate multiple simple events. Thus, the number of events decreases with every process step. The definition for Complex Event Processing is:

Definition 4. COMPLEX EVENT PROCESSING (CEP): Computing that performs operations on complex events, including reading, creating, transforming, or abstracting them.

The difference between definitions [4] and [3] is that the operation is based on complex events, instead of simple ones. A complex event is an abstraction of other events, called its members. Hence, only one event is necessary to represent these events which reduce the number of events. There are different types of complex events, like derived event and composite event. A derived event is generated as a result of applying a method or process to one or more other events. A composite event is a derived, complex event that is created

by combining base events using a specific set of events constructors such as disjunction conjunction, sequence etc. The term aggregate event is sometimes used for some forms of composite or derived event.

In the baggage management system an event occurs if a bag reaches its destination airplane. If all baggage is on board a complex event is generated to signal that the airplane can start. After this, only the complex event is important and the input events can be deleted under the precondition that the events are not required by other operations.

Events, simple as well as complex ones, are processed on the basis of rules. A rule is a prescribed method to processing events. The input of a rule can be multiple different types of events. The generated result event can be input for other rules.

In CEP, the event sources are often event streams. An event stream is a “linearly ordered sequence of events” [LS08]. For instance, if a sensor measures every second the new value, then every measurement is send as event in a stream. Hence, event queries have to evaluate continuously during events occur. Processing events of an event stream is called Event Stream Processing (ESP)

There are several languages providing event queries on the market. In [EB09] are defined four requirements for such languages:

- *Data extraction*: It should be possible to extract data from the event message
- *Composition*: Several events are summarized in one complex event
- *Temporal Relationship*: It should be possible to define temporal conditions as well as causal ones
- *Accumulation*: Queries are defined on a certain window of the stream, otherwise negation as absent of an event cannot be evaluated.

Complex events are processed in software module called CEP-Engine which performs multiple rules. The CEP-Engine can act as event source as well as event sink. Due to that it can be build a network, where the nodes are CEP-Engines and the edges are event streams. Such a network is shown at Figure 1.2 in the introduction. The events are getting a hierarchical structure. At the beginning there are lots events, during they are processed in the network, the number decrease but the expressiveness of each event are increased.

2.2.1 Esper

In this work Esper is used as CEP Engine. Esper is open-source and written entirely in Java. Esper reverses the acting by a database system. Instead of storing the data and then run queries on it, Esper allows application to store queries and run the data through them. The complete documentation is given in [Esp12]

Esper enables to formulate continuous queries easily, because it provides an SQL-like language extended to handle event streams and event pattern called Event Processing Language (EPL).

Esper features an expressive Event Processing Language

- Continuous queries, filtering and computations (min, max, avg, stddev, vwap, ...)
- Continuous joins
- Followed-by logic, repeat patterns and detection of missing events
- ESP and CEP federated processing
- Time and length based sliding window
- Output flow control
- Continuous joins of streams and historical data stored in relational databases, with local caching
- High performance, low latency

In Figure 2.2 the processing model is shown.

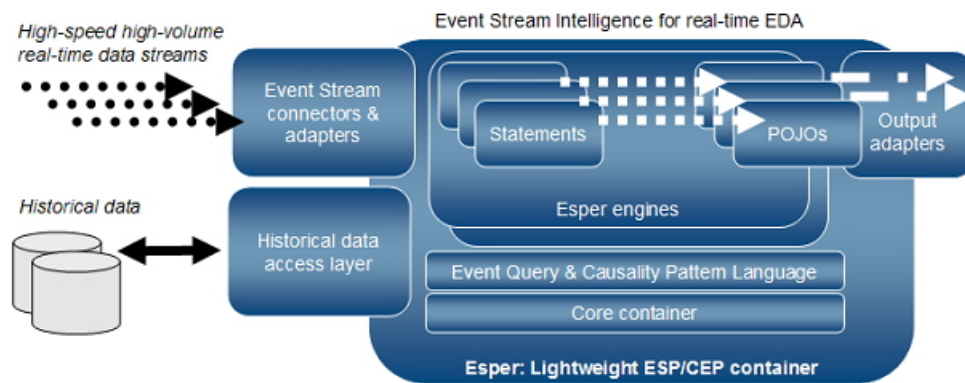


Figure 2.2: Esper Process Model [Esp08]

At the beginning event queries and pattern statements are registered in the Esper container. At the execution time the events flow through the event stream connectors and adapter. Afterward arbitrary logic is triggered and bound to the engine in the form of standard java objects.

Esper offers different event representation as Java objects, .Net objects, Map or SML as well as several input and output adapter for CSV, JMS, DB, HTTP and Sockets. This enables embedding event processing in existing Java applications or middleware to add real-time event- driven capabilities to existing platforms without paying high serialization cost or network latency for every message received and action triggered. Esper provides two principle methods or mechanisms to process events: event pattern and event stream queries.

The event stream queries containing the same language constructs as SQL: SELECT, FROM, WHERE, GROUP BY, HAVING and ORDER BY. Listing 2.1 shows an example: This query returns as result all event messages from Sensor B. In event stream processing streams replace database

tables as the source of data with events and replacing tables rows as the basic unit of data. Since events are composed of data, the SQL concepts of correlation as joins, filtering and aggregation, makes it possible that grouping can be effectively leveraged. EPL statements are used to derive and aggregate information from one or more streams of events, and to join or merge events streams. An important feature is the possibility to define one or more views, which make the data available for querying and filtering. Some views represent windows over a stream of events as shown in Listing 2.2: This query returns as results all event message, that are sent by the sensor B in the last 10 seconds. Other views derive statistics from event properties group events or handle unique event property values. The Esper engine ensures that views are reused among EPL statements for efficiency. EPL queries are created and stored in the engine. The results are published to listeners as events. The implemented listener specifies the reaction of the result. For example, a new event can be created with the result information, that are input for another query.

```
SELECT *
FROM Sensor_Events
WHERE Sensor_Name = B
```

Listing 2.1: Esper Query

```
SELECT *
FROM Sensor_events.win:time(10 sec)
WHERE Sensor_Name = B
```

Listing 2.2: Esper query with window

Event pattern languages specify expression-based event pattern matching. An event pattern matches if an event or multiple events occur that match the pattern's definition. Pattern expression consists of pattern atoms and pattern operators. Pattern atoms are the basic building blocks of patterns. Pattern operators control expression lifecycle and combine atoms. There are four kinds of operators: operators that are control pattern sub-expression repetition (every, every-distinct and until), local operators (and or not), temporal operators that operate on event order (followed by) and guards that are where-conditions controlling the lifecycle of subexpressions (timer, within, max and while expression). A Pattern can be anywhere in the FROM-clause of an EPL statement including joins and subqueries. Patterns maybe used in combination with the WHERE, GROUP BY, HAVING clauses as well as to limit the output rate. Example 2.3 shows a pattern. The result of this query is all events where the message from A arrives directly before B.

```
SELECT *
FROM PATTERN [every a=sensor_event(name=a) every b=sensor_event(name=b)]
```

Listing 2.3: Esper Pattern

2.3 Cloud Computing

This chapter gives an overview about Cloud Computing. William Voorsluys et al. [VBB11] describe the roots of Cloud Computing as the advancement of several technologies, especially hardware, internet technology, distributed computing and system management. Cloud Computing has caused a change from in-house generated computing power into utility-supplied computing resources delivered over the internet as Web Service. Both, the provider and the consumer, have benefits. On the consumer side the IT-related cost are reduced by choosing cheaper service from a provider and has not to investigate in IT-infrastructure and personnel hiring. Providers achieve better operational cost. Hardware and software infrastructures are built to provide multiple solutions and serve many users. Therefore, the efficiency is increased. The time of return on investment is reduced as well as the total cost of ownership.

There are lots of different vendors offering Cloud services, but there are just a few standards on this area. Hence, there exists no general definition of Cloud Computing. However, the National Institute of Standards and Technology (NIST) has given some characteristics which are essential for a Cloud [MG11].

- *On demand self-service*: A consumer can provision resources without requiring human interaction with the provider.
- *Broad network access*: Capabilities are available over the network and can be accessed through standard mechanisms.
- *Resource pooling*: The provider's computing resources are pooled and can serve multiple consumers based on a multi-tenant model. Different physical and virtual resources dynamically assigned and reassigned according to consumer demand. The consumer does not know the exact location of the provided resources but sometimes on a higher abstraction level as country or state.
- *Rapid elasticity*: Resources can be rapidly and elastically provisioned. Sometimes, the scale is done automatically. The available capabilities for provisioning, often appear to the consumer as be unlimited and can be purchased in any quantity at any time
- *Measured Service*: Cloud systems automatically control and optimize the use of the resources by leveraging a metering capability at some level of abstraction appropriate to the type of service. Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service

There are different abstraction levels to offer capabilities for the provider. The three classes of cloud computing service level are shown in Figure 2.3: Infrastructure-as-a-Service (IaaS), Platform-as-a-service (PaaS) and Software-as-a-Service (SaaS).

[MG11] defined the Service Models as following:

Software-as-a-Service: Provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through a thin client interface such as a web browser. The consumer has no access to the Cloud infrastructure.

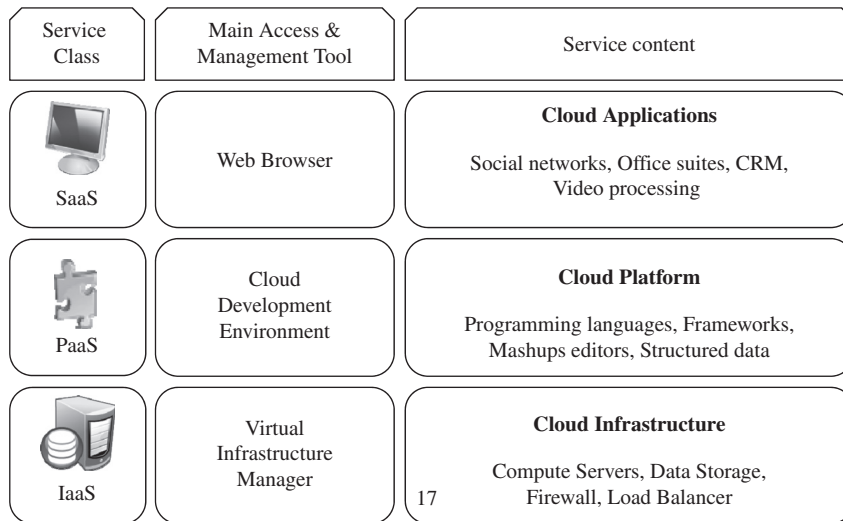


Figure 2.3: The Cloud Computing Stack [VBB11]

Platform-as-a-Service: The consumer can deploy consumer-created or acquired applications onto the Cloud infrastructure using programming languages and tools supported by the provider. The access to the Cloud infrastructure is limited,. The consumer has only control over the deployed applications and possibly applications hosting environment configurations.

Infrastructure-as-a-Service: The provider offers virtualized processing, storage, network, and other fundamental resources. The consumer can run arbitrary software on that, including the operating system. Therefore, he has control over the operating systems, storage, deployed applications and possibly limited control of networking component, but not on the underlying Cloud infrastructure.

Cloud computing do not only differ in its offered services, but also on the environment which the services are deployed. In [MG11] are defined four different kinds of cloud-based deployment models are defined: Private Cloud, Community Cloud, Public Cloud and Hybrid Cloud.

Private Cloud: The Cloud infrastructure is operated solely for an organization. It may be managed by the organization or a third party and may exist on premise or off premise.

Community Cloud: The cloud infrastructure is shared by several organizations and supports a specific community that has shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be managed by the organizations or a third party and may exist on premise or off premise.

Public Cloud: The Cloud infrastructure is made available to the general public or a large industry group and is owned by an organization selling Cloud services.

Hybrid Cloud: The Cloud infrastructure is a composition of two or more clouds (private, community, or public) that remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds)."

A private cloud is often a data center which is extended by virtualization and cloud-like interfaces [VBB11]. Due to this, security and privacy is better than in a public cloud, where multiple users share the same resources. However, the cost of a private cloud is constant. It does not matter, if the resources are used or not. As opposed to the data center, the resource can be more utilized. The big advantage of public cloud is the "pay-as-you-go"-manner, this reduces the running cost. The problem is that Cloud provider typically offers only standardized sizes of the resources as small, medium and large. Thus, the resources are not optimized on the consumer requirements.

The following section describes more precisely IaaS, because in this work we will focus on that layer.

2.3.1 Infrastructure-As-A-Service

As mentioned in the introduction virtualization is an important topic in Cloud Computing and one key feature of IaaS. To make that clear, it is just needed to look at the past. It was very difficult to install a new server for a particular task. The IT-administrator had to investigate a lot of time going through the whole checklist doing the specific procedure. With the virtualization technology involved in Cloud Computing it takes quite fewer time to do the same. The consumer has only to define the properties of the virtual server and provision it over a self-service interface. In shortest possible time, the new server with all requirements is available. [ER11]

As we can see in the example, virtualization allows allocating new resource easily. Additional, the flexibility increased, because virtual machine can be scaled up and down on demand and on consumer side it seems, that there is an infinite availability of resources.

In Figure 2.4 the lifecycle of a virtual machine in a Cloud is shown. With that it obvious, that no immediately after sending the allocation request the new virtual machine is available for the user. If the allocation request is received by the Cloud provider, the IT administrator has to find a good place for the virtual machine according to its requirements. The operating system and the applications has to be load as well as customizing and configuring the virtual machine. Then the server starts and the virtual machines is available by a web service for consumer. During the operation, services can be migrate and the compute resources can be scale. After the deallocation request service is ended and the compute resources deallocate to other virtual machines.

An example for a Public Cloud providing IaaS is the Amazon Elastic Compute Cloud (EC2). Users can provision new virtual machines into the amazon's virtualized infrastructure through a web service. Additional features are rebooting their instances remotely, scaling

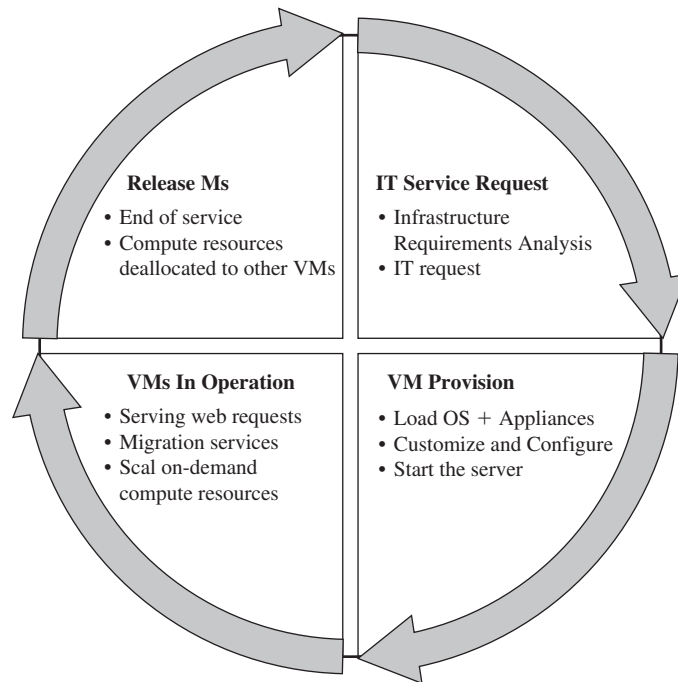


Figure 2.4: The lifecycle of a virtual machine [ER11]

capacity quickly and using on-demand service when needed. They get full root access and can install almost any OS or application in their machines. Amazon offers 6 different types of instances: Standard Instances (Sizes: Small, Large, Extra Large), Micro Instances, High-Memory-Instances (Sizes: Extra Large, Double Extra Large, Quadruple Extra Large), high-CPU Instances (Medium, Extra Large), Cluster Compute and Cluster CPU. They differ at the rate of CPU-performance and memory.

Related Work

As pointed out in the introduction a main problem in CEP are the peaks in the workload, where more and more events have to process. There are several approaches to smooth the workload by load balancing. The simplest one is load shedding. Load shedding is dropping events in order to decrease the load. The problem of load shedding is that omitting events leads to less accuracy in the result. Tabul et al. [TÇZ⁺03] developed various techniques for inserting dynamically drop operators. They specify Quality of Service (QoS) rules in order to determine events having the less impact on the accuracy of the result. These events are candidates for dropping.

Another approach without load shedding offer distributed CEP systems, because there are multiple node performing various rules. Load balancing can be done by migrating rules between different nodes. Nodes with a high working load migrate rules to other nodes with less workload. Cherniack et al. discuss in [MHM⁺03] the architectural issues to extent their existing centralized stream processing Aurora, shown in Figure 3.1, into a distributed one. The focus is on load management, high availability and federate operation issues. Aurora* is a network consisting of several Aurora nodes, which acts like a centralized streaming processor. Each node monitors its local operation, its workload and available resources. If more resources are needed, it will consider offloading rules to another appropriate Aurora node. They introduce two techniques box sliding and box splitting to distribute rules. One

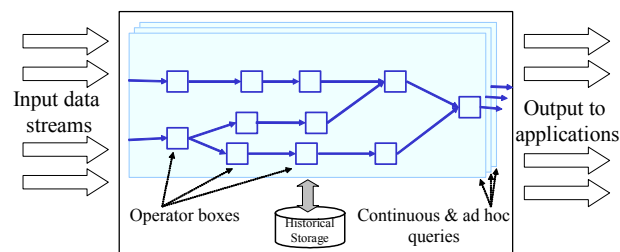


Figure 3.1: Aurora system model [MHM⁺03]

problem obtained, there are limited resources. If every node reach its limit, no ruler can be migrate and the nodes overload.

Therefore, Kleiminger et al. introduce in [KKP11] a combined stream processing system. Events are processed locally as well as in the Cloud. The advantage for processing events in the Cloud is the seeming unlimited availability of resources. As shown in Figure 3.2, there are a centralized load balancer, a local stream processor and a Cloud stream processor. The local stream processor handles the average load and the Cloud stream processor is responsible for the peaks in the workload. The load balancer monitors the state of the input queue and decides on this state which processor is used. The state is depended on the size of the queue. If a specified size is reached, not only the local stream processor processes event but also the Cloud processor. The problem is, that a centralized load balancer is not scalable.

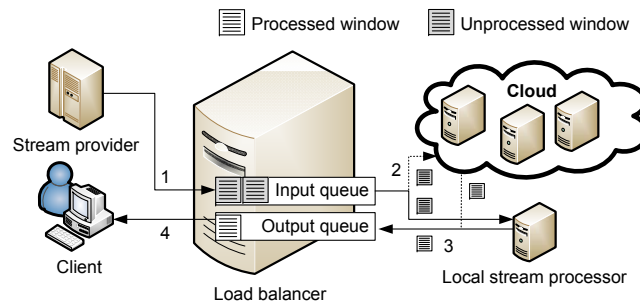


Figure 3.2: Kleiminger's approach [KKP11]

Randels et. al in [RLTB10] recognized that a centralized load balancer is a bottle neck in distributed system. Therefore, they present a distributed load balancing approach on the basis of natural inspired algorithms for the Cloud. Three methods are introduced in order to maximize the global goal by local node interaction. Their goal is to make the best use of the resources. The methods are honeybee, random sampling and active clustering. For example, the honeybee is a biologically inspired searching algorithm that mimics the food foraging behaviour of honey bees swarms. They try find the best feeding place for the swarm by the local search of each bee. For our application good food are resources that have high effective usage. Resource usage is measured locally by the profit function. Hence, unprofitable resources are abandoned and profitable ones are allocated more task. The approach is developed for balancing load in the cloud but maybe, it should apply on load balancing in CEP systems as well.

Hummer et al. [HLS11] have introduced a method for dynamic migration of processing elements between different nodes. This method is developed for WS-Aggregation. The processing elements are event buffer, preprocessor and queries including the subscribing metadata. An example is shown in Figure 3.2. An aggregator node can execute multiple queries. An event buffer has to be duplicated if two queries located on different nodes use the same input data source. During runtime queries can be added and deleted on the system.

Additionally, queries can be migrate from one node to another one for load balancing. The migration method does not only respect on load balancing, the look as well for avoiding duplicating buffers and minimizing the data transfer between nodes. Therefore, the method tries to find a valid solution for the query placement under the given constraints. The load of each node is determine by the stress level provided by a metadata interface of the Cloud node. If the stress level is at the maximum, queries should be migrate to another node. If the algorithm can not found a valid solution, an new machine is requested from the cloud environment and the optimization is restarted.

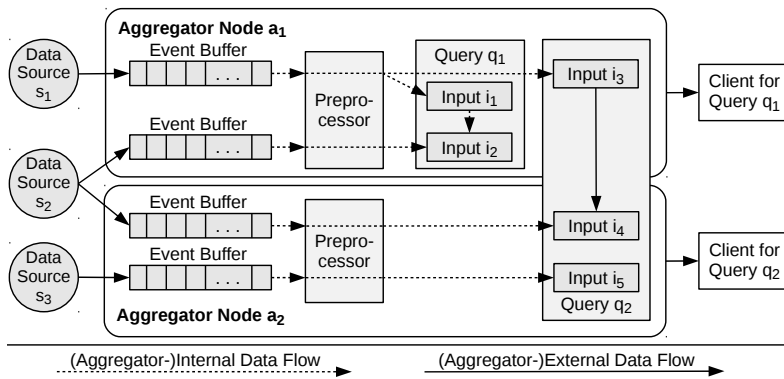


Figure 3.3: Hummer's approach [HLS11]

The System Model

The following section describes the system model developed for CEP in the Cloud.

4.1 The CEP Network

A CEP application is modeled by a rule network. The rule network in standard distributed CEP Systems is mapped to a physical network by deploying the rules on physical nodes. Often, multiple rules are deployed on one physical node. An example is given in Figure 4.1. In this work the rule network is mapped to a virtual network. The virtual environment allows to deploy every rule on a single virtual machine provided by the cloud as shown in Figure 4.2.

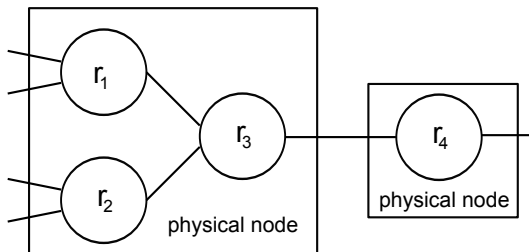


Figure 4.1: Old mapping

4.1.1 Eventrate

The eventrate defines the number of incoming events per seconds. The incoming events have to be processed by a specific rule. In order to determine the eventrate, the incoming events per second are counted. The measurement unit is events per second (e/s). For instance, if the eventrate is $100e/s$, then 100 event arrived within the last second.

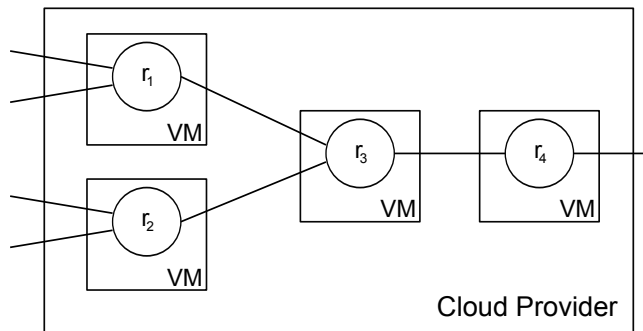


Figure 4.2: New mapping

The eventrate is not constant over the time. There are peaks with more incoming events and minima with smaller number of events. The function $rate(t) : time \rightarrow eventrate$ defines the change of the eventrate. An example is given in Figure 4.3.

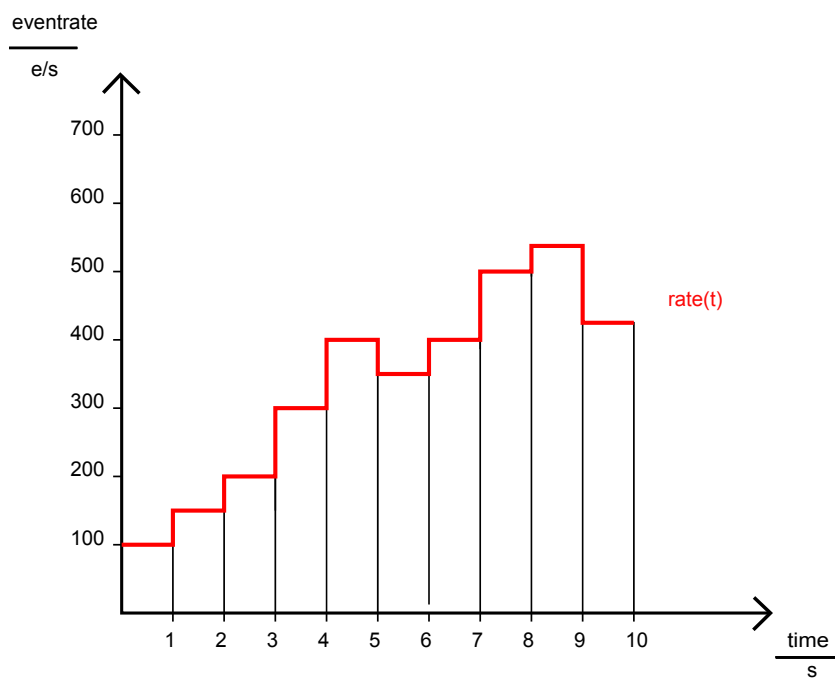


Figure 4.3: The-function-rate()

The x-axis is the time dimension and the y-axis is the eventrate. For example between second 2 and 3, there are 300 events being processed. One second later, there are 400 events. Thus, the eventrate changes.

4.1.2 Virtual Node

A virtual node v_i is a virtual machine which performs rule r_i . Each virtual machine has the same fixed size of maximum computing power. The larger the eventrate is the more computing power is needed. Furthermore, the higher the complexity of the rules is the more computing power is needed [Ste]. Due to the fixed maximum computing power of each node and since one node performs exactly one rule, for each node v_i a maximum eventrate e_{max} can be defined as the eventrate which effects that the workload of v_i is 100% and, simultaneously, no events are stored in the queue. We assume, that all virtual machines have the same maximum computer power and therefore, e_{max} is equal for all virtual nodes deploying the same rule. If two node perform different rules, e_{max} can vary.

In order to process more events than e_{max} , a new node with the same rule can be allocated. The two nodes share the workload.

4.1.3 Event Stream Splitting

In order to distribute the workload to multiple nodes, the input event stream has to split. In general, this is not easy, because events are not independent from each other and some rules need the event stream history to get the result. For example, a rule should be recognizing, if the event A is directly follow by event B. The result is wrong, if event A is passed to another node as B. The solution of this problem is not in the focus of this work. There are related works, which have already considered this problem [BKS⁺99, LRE09]. In the following, we assume that well-performing splitting components are already available.

4.1.4 Rule Cluster

All nodes v_i performing the same rule r_i belong to the same cluster C_i :

$$C_i = \{v_i, v'_i, v''_i, v'''_i, \dots\} \quad (4.1.1)$$

The first node v_i in a cluster is the major node, which cannot be deleted and the resource manager is located on it. The function of the resource manager is explained in the next section. Every cluster has a branch and a merge component. The branch component divides the incoming event stream equally to the node within the cluster. An example is given 4.4.

The function $rate_{r_i}()$ returns the number events, which have to be processed by the rule r_i . The input of node v_i is

$$rate_{v_i} = rate_{r_i}(t) / |C_i| \quad (4.1.2)$$

The merge component merges all output events into one stream.

Every node within a cluster has the same e_{max} , because all perform the same rule. Therefore, the maximum eventrate of a cluster is is:

$$e_{max}^{C_i} = e_{max}^{v_i^1} + e_{max}^{v_i^2} + e_{max}^{v_i^3} + \dots \quad (4.1.3)$$

It follows, the more nodes are in the cluster, the more events can be processed without delay.

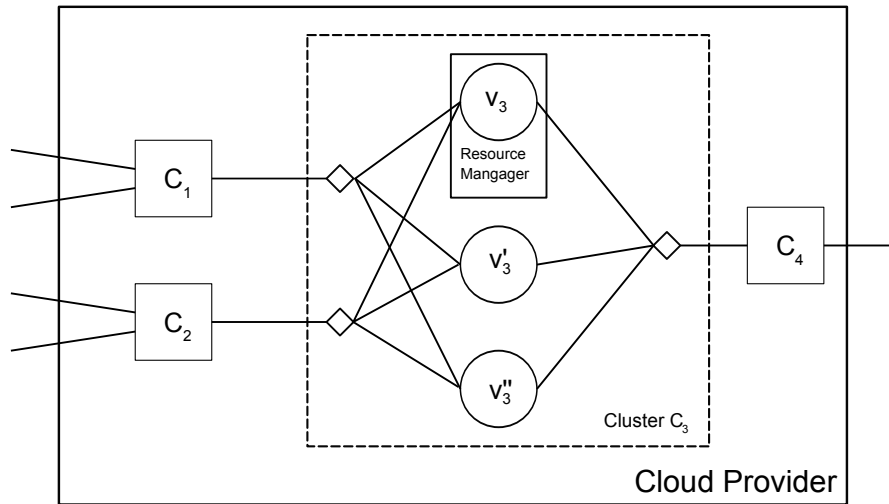


Figure 4.4: Cluster

4.2 Cloud Provider

For our approach a cloud provider should be used, that offers IAAS. Therefore, it is possible to allocate a new node on demand as well as deleting an existing one. One inconvenience is that the new node is not available directly after the request. According to this, the following parameter are defined

- t_a : Point of time of the allocation request
- t_b : Point of time, when the node is available for the customer
- λ : Time between request and node is available

This leads to the following equation:

$$t_a + \lambda = t_b \quad (4.2.1)$$

The value of λ is given by the Cloud Provider and differs from Cloud Provider to Cloud Provider.

Chapter 5

Resource Manager

The resource manager, located on each major node of a cluster, has to ensure that a cluster has enough computing power, that no event is stored in the queue. This is the case as long as the maximum eventrate of the cluster is higher than the incoming events:

$$e_{max}^{C_i}(t) \geq rate_{r_i}(t) \quad (5.0.1)$$

Additionally, the resource manager arrange that as less as possible resources are allocated. This is the case as long as inequation 5.0.1 is valid and if the total eventrate of the cluster with one node less is smaller than the incoming events:

$$e_{max}^{C_i}(t) - e_{max}^{n_i} < rate_{r_i}(t) \quad (5.0.2)$$

Once equation 5.0.1 is false, that means the eventrate is greater than e_{max} , we have to allocate additional resources. We denote this as critical point in time as t_c . The resource manager has to send an allocation request to the Cloud Provider in order to add a new node to the cluster, when the inequation 5.0.1 is not valid. Otherwise, if the equation 5.0.2 is not valid, it has to send a deallocation request. The challenge is to determine the point of time t_a , when the resource manager has to send the request. For the allocation request the following has to be considered.

At time t_c a new node has to be added. This is the case when for the first time the incoming events are more than the maximum eventrate of the cluster:

$$e_{max}^{C_i}(t) > rate_{r_i}(t) \quad (5.0.3)$$

However, the new virtual machine is provided by the cloud provider. Therefore, the equation 4.2.1 has to be considered. The time t_b is the time, when the new node is available for the customer. Thus, this should be equal to t_c . Then the maximum eventrate is decrease and the equation 5.0.1 is valid. For t_a follows:

$$t_a = t_c - \lambda \quad (5.0.4)$$

If the new node is available at time t_c , $e_{max}^{C_i}$ is increased and the equation 5.0.1 is valid. The request has to be sent before the critical time is reached. The problem is, that $rate()$ is not given in advanced. Thus, time t_c has to be estimate and as a consequence t_a . Therefore, it can happen that t_c and t_b are different. The goal should be to minimize the difference Δt_{error} :

$$\Delta t_{error} = t_b - t_c \quad (5.0.5)$$

A criterion to determine t_a is introduced in the next section.

A deallocation request can be sent if the maximum processable eventrate of the cluster is still greater than the incoming events, if one cluster node is dropped

$$e_{max}^{C_i}(t) - e_{max}^{n_i} > rate_{r_i}(t) \quad (5.0.6)$$

Immediately after sending the request the node is no longer available. It is desirable, that the deallocation request is not send at the first time, when equation 5.0.6 is valid, because otherwise is can happen, if the incoming events varies only by a few events, every time a allocation or deallocation requests is sent. Therefore, the difference between the maximum eventrate with one node less and the incoming events should be greater than a specific constant:

$$e_{max}^{C_i}(t) - e_{max}^{n_i} - rate_{r_i}(t) > const. \quad (5.0.7)$$

5.1 Decision Criterion: Critical Workload

In our approach, the criterion in order to decide the time of sending the request is to define a critical workload. The critical workload specifies the state of a cluster, where it is highly probable that a new node is need soon in order to handle the increasing workload. Thus, at this state a new node has to be allocated. It is defined by threshold h and the maximum eventrate of the cluster.

$$h * e_{max}^{C_i}$$

The value h is between 0 and 1. Therefore, the critical workload is between $h * e_{max}^{C_i}$ and $e_{max}^{C_i}$. A allocation request has to be sent if at the first time the incoming eventrate is greater than the threshold:

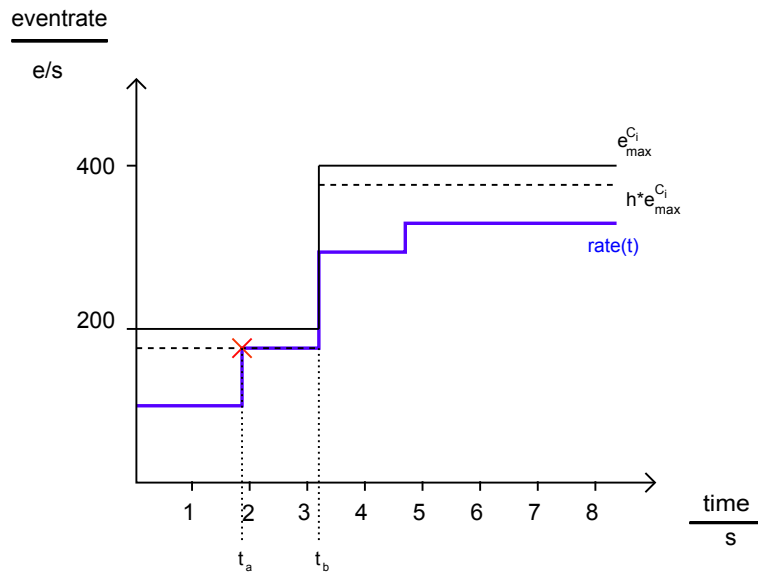
$$h * e_{max}^{C_i}(t) < rate_{r_i}(t) \quad (5.1.1)$$

The higher h is chosen, the later the request is send. It has to be ensure, that only once the allocation request is sent as long as the rate is in the same critical workload. An example is shown in figure 5.1. .

The critical workload should be also considered by sending a deallocation request in order that equation 5.0.7 is valid. It follows, the request should be sent if the incoming event rate is smaller than the critical workload from the cluster with one node less

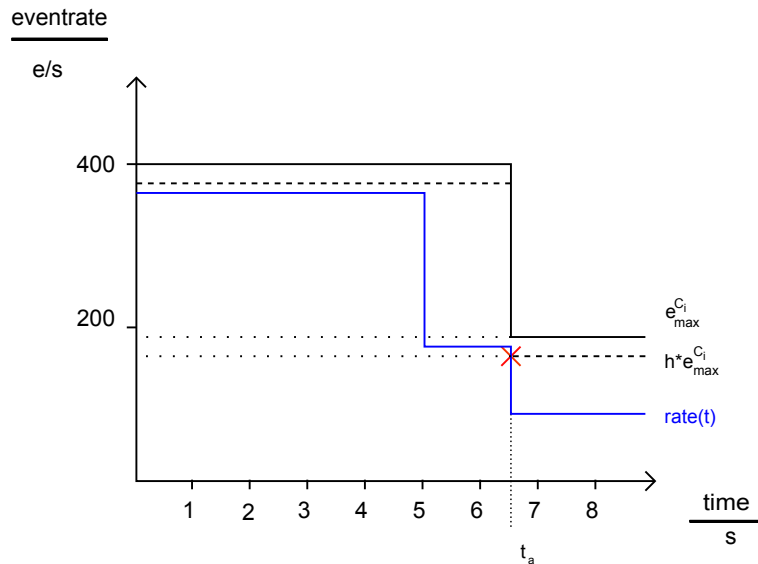
$$h * (e_{max}^{C_i}(t_a) - e_{max}^{n_i}) > rate_{r_i}(t) \quad (5.1.2)$$

With this equation it is avoided that the next critical workload is reached and a allocation request is sent. An example is given in Figure 5.2.



The threshold is reached at t_a . Thus, the allocation request is send. At time t_b the new node is available. Therefore, the maximum eventrate is increase and can processed more events without delay.

Figure 5.1: Allocation process



The deallocation request is sent after $rate()$ is smaller than the threshold. If the request is sent after 5s, $rate()$ will be in the next critical workload and a allocation request will be sent.

Figure 5.2: Deallocation process

Evaluation

In this chapter the introduced threshold criterion is evaluated. Therefore, the correlation between the threshold and the function rate and the impact on the error is considered.

6.1 Implementation

To evaluate the suggest criterion a test environment was developed in Esper. In the test environment, there are two components: the event generator and the major node. The event generator produces events and sends them to the major node that performs a rule and runs the resource manager.

The event generator sends every second a specified event number to the major node. To test the behavior of an increasing eventrate, the number of events is increased every second by f , which specifies the number of the additional events per second. For example, if the increasing factor is 100, then at the first second 100 events are sent and at the next second 200 events and the next second 300 and so on. The value of f has impact on the function $rate()$. The higher f is chosen the larger is the gradient of the function $rate()$. In addition, the event generator splits the event stream to the several nodes. Therefore, it stores the number of the virtual machines. On the basis of this number, the event generator sends only the correspondent proportion to the major node. For instance, if there are two virtual machines and 400 events should be processed, the event generator sends 200 events to the major node.

The major node performs the rule presented in 6.1. The rule produces every second a result counting the incoming events. On that rule an update listener is installed, which is called, if a new result is produced. The update listener contains the resource manager, if the counting result is greater than the specified threshold h , then a new virtual machine is allocated by sending a request to the event generator. It is ensured that only one request is send for the same critical workload.

```
SELECT count(*)  
FROM Incoming_Events.win:time_batch(1 sec)
```

Listing 6.1: Example

6.2 Error description

As mentioned in the chapter before (5.0.5), the goal is to minimize the difference (Δt_{error}) between the time when a new node is available (t_b) and the time when the node is needed (t_c).

There are two different possibilities that t_{error} arise.

1. The new node is available before it is needed:

$$t_c > t_b \quad (6.2.1)$$

This mean, for t_{error} seconds, more resources are allocated than needed. Hence, the equation 5.0.2 is not valid.

2. The new node is later available as it is needed

$$t_c < t_b \quad (6.2.2)$$

This means, for Δt_{error} seconds, events are stored in the queue. Hence, the equation 5.0.1 is not valid.

In order to compare the different results a mapping between t_{error} and the events should be done. Therefore, the number of affected events of the error is calculated. In case 6.2.1, this are the events which can be potential processed more. In opposed to case 6.2.2, that are the events which are stored in the queue. The number of the affected events can be calculated through the integral of $rate()$. The boundaries are t_c and t_b .

$$E_{error} = \int_{t_1}^{t_2} rate(t) - e_{max}^{C_i}(t) \quad (6.2.3)$$

$t_1 = t_c$ and $t_2 = t_b$ for 6.2.2 otherwise $t_1 = t_b$ and $t_2 = t_c$

As shown in Figure 6.1, in the test cases the $rate()$ is linear function, therefore the integral is a rectangle triangle. The area can be calculated by $A = \frac{1}{2} * a * b$, where a and b are the catheti of the rectangle triangle.

Through the measurement we get following results

$$rate(t_1) = r_1$$

$$rate(t_2) = r_2$$

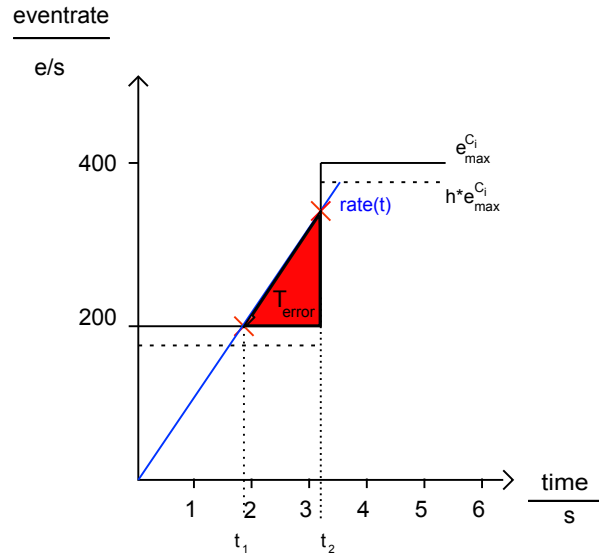


Figure 6.1: Error Area

It follows, that the area is

$$E_{error} = \frac{1}{2} * (t_2 - t_1) * (r_2 - r_1) \quad (6.2.4)$$

6.3 Test Series

The correlation between the threshold and the function $rate()$ should be investigated focusing as the impact on t_{error} . In the first test series, the number of the additional events f is different in each run. In test series two the threshold h is different. In the last case a long term observation is done.

In every test, the e_{max} of one virtual node is $1000e/s$. The delay λ of the cloud provider is $2s$.

Every second the actual eventrate $rate()$, the maximum evenrate of the cluster $e_{max}^{C_i}$ and the threshold workload $h * e_{max}^{C_i}$ are recorded. The results are shown in the respective figures.

6.3.1 Test Series 1: Different Gradients

In the first test series the gradient of the function $rate()$ are different, each run has a different number of the additional events f . The threshold h is $0,8$.

Following tests are conducted:

Test a) $f=100$

Test b) $f=30$

Test c) $f=200$

Test a) $f = 100$

The first test the number of the additional events f of 100. The result is shown in Figure 6.2. At t_1 the allocation request is sent. At time t_2 the two function $e_{max}^{C_i}$ and $rate()$ has a interception point and therefore t_{error} is zero as well as E_{error} . Hence, the new node is exactly available when it is needed and no events have to be stored. It follows the relation between $rate()$ and threshold h is perfect.

Test b) $f = 30$

In the next run, f is 30. In Figure 6.3 the effect is shown. At time $t_1(28s)$ the request is sent, the new node is available at $t_2(30s)$, but the new node is needed at time $t_3(35s)$. It follows that $\Delta t_{error} = 5s$ and 6.2.1 is the kind of error. The E_{error} can be calculated by

$$E_{error} = \frac{1}{2} * (t_3 - t_2) * (r_3 - r_2) = \frac{1}{2} * (35s - 30s) * (1000e/s - 900e/s) = 250e$$

Hence, the 250 events could be processed more.

Test c) $f = 200$

In the next the number of the additional events f is 200. As shown in Figure 6.4 at time $t_1(5s)$ the allocation request is sent. A new node is needed at $t_2(6s)$, but the new node is not available until $t_3(7s)$. It follows $\Delta t_{error} = 1$. At time t_3 should be processed 1200 events, only 1000 can be processed, because of the maximum eventrate. Therefore, 200 events are stored in the queue and are processed in the next period The kind of the error is 6.2.2. The E_{error} can be calculated by

$$E_{error} = \frac{1}{2} * (t_3 - t_2) * (r_3 - r_2) = \frac{1}{2} * (7s - 6s) * (1200e/s - 1000e/s) = 100e$$

Hence, 100 events have to stored in the queue.

Result

Based on the perfect relation as in test a), it follows: The flatter the gradient of the function rate, the larger is the error of kind 6.2.1. The steeper the derriation of the function rate() the larger is the error of kind 6.2.2.

In summary, the gradient has impact on the error.

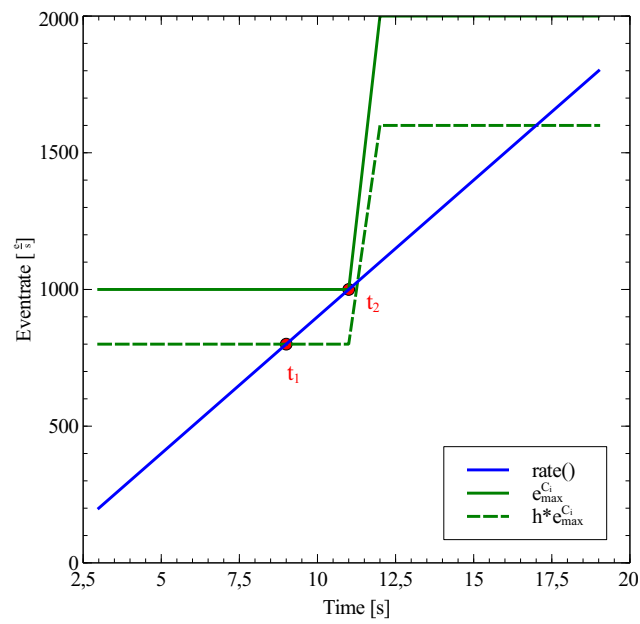


Figure 6.2: Test b: Increasing 100 Events per second

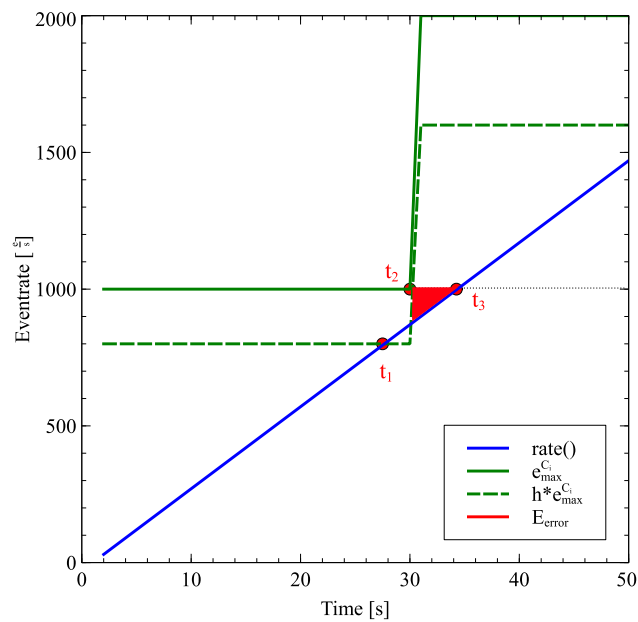


Figure 6.3: Increasing 30 Events per second

6.3.2 Test Series 2: Different Thresholds

According to the perfect case shown in Figure 6.2 with a number of the additional events $f = 100$ and a threshold $h = 0.8$, in the following test series the the increasing factor is fix and the threshold is varied. Two test has been run:

Test a) $h = 0.9$

Test b) $h = 0.7$

Test a) $h = 0.9$

Figure 6.5 shows the results of test a). At $t_2(12s)$ the new node is needed, because the maximum eventrate and the rate() has an interception point, but eventually the new node is available at $t_3(13s)$. It follows that $\Delta t_{error} = 1s$. Case 6.2.2 arises. The E_{error} can be calculated by

$$E_{error} = \frac{1}{2} * (t_3 - t_2) * (r_3 - r_2) = \frac{1}{2} * (13s - 12s) * (1100e/s - 1000e/s) = 50e$$

Test b) $h = 0.7$

In test b), the threshold is 0.7. This results in case 6.2.1, shown is figure 6.6. At $t_2(10)$ the new node is available and at $t_3(11s)$ the function rate() has reach the maximum eventrate.

$$E_{error} = \frac{1}{2} * (t_3 - t_2) * (r_3 - r_2) = \frac{1}{2} * (11s - 10s) * (1000e/s - 900e/s) = 50e$$

Result

Based on the perfect relation between the function rate() and the threshold as shown in Figure6.2, it follows: The smaller the threshold is the greater the error of kind 6.2.1 is. The greater the threshold is the greater is the error of kind 6.2.2.

In summary, the threshold has impact on the error.

6.3.3 Test 3: Long-term observation

In this test, the long-term behavior of the correlation between rate() and threshold is investigated. Therefore, the best case parameters are chosen: $h = 0.8$ and $f = 100$. As we can see in Figure 6.7, at t_1 the function rate() and the maximum evenrate has a interception point and, therefore, Δt_{error} is zero. With every additional virtual machine t_{error} is increased.

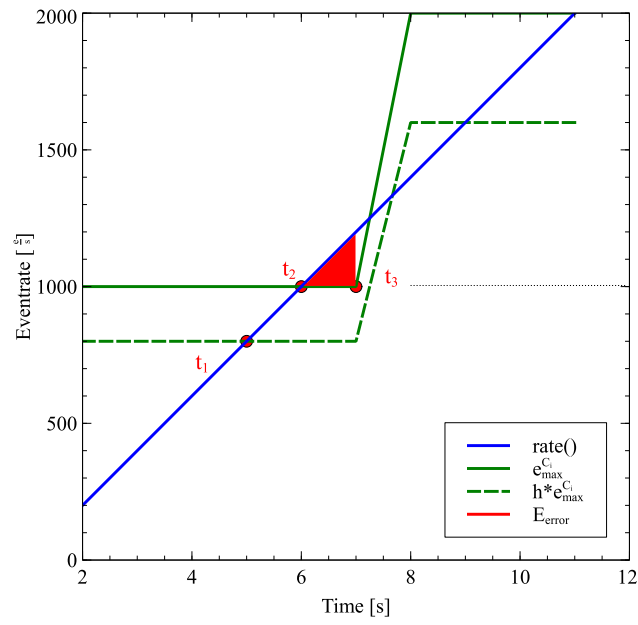


Figure 6.4: Increasing 200 per second

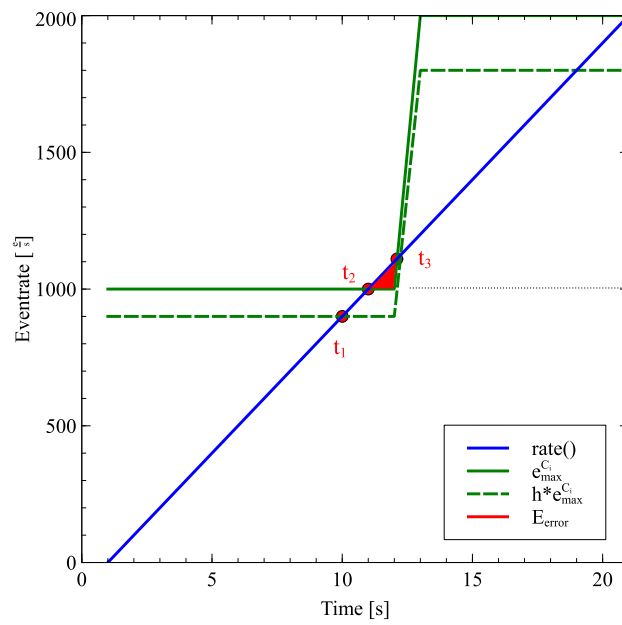


Figure 6.5: Test a: Threshold = 0,9

At time $t(35s)$ the new virtual machine is available, but at $t_3(41s)$ the new node is recently needed. t_{error} is 6s. Additionally, the next threshold is reached, and a new allocation request is sent. The problem is, that the threshold is fix and the critical workload is increasing, the more virtual machine is in the cluster.

$$E_{error} = \frac{1}{2} * (t_3 - t_2) * (r_3 - r_2) = \frac{1}{2} * (41s - 35s) * (4000e/s - 3200e/s) = 2800e$$

The value of E_{error} shows, that there are more computing power available to process other events. The different between $rate(t_1)$ and $rate(t_2)$ is $800e/s$ this value is very high according to the fact that the maximum eventrate of the node is $1000e/s$.

It follows, that the threshold should be adaptable on the number of virtual machines.

6.3.4 Conclusion

As we can see, the threshold and the behavior of $rate()$ influence Δt_{error} as well as E_{error} . There are two different kinds for arising an error, both are dependent on the relation between $rate()$ and the threshold. It follows, the choose of the threshold is correlated to the $rate()$ function. In addition, the provisioning time λ of the Cloud provider has also impact on the error, through the behaviour of the value e_{max}^i . The greater λ is the greater the error of case 6.2.2 is. As opposed to case 6.2.1 where the error is reduced by a high provisioning time. However, provisioning time of a Cloud provider is fix for all nodes and can be compensate by the correct choose of the relation between $rate()$ and the threshold.

For a weaker increasing eventrate the threshold can be chosen smaller than for a rapidly increasing evenrate. The more prior knowledge about the function $rate()$ is available, especially its gradient, the better the threshold can be chosen. Note, that the threshold of different clusters can be varies, because each cluster can have a different eventrate.

Furthermore, it should also be consider, which kind of error can the application better handle. If it is important, that no events are delayed, the threshold should be chosen smaller than if it is important. On the other hand, if the cost of the resources should be optimized than the threshold should be higher.

As the long term observation shows, it is not good if the value of the threshold is fixed, because the more virtual machine are exist, the greater is the critical workload and the greater is t_{error} . A better solution can be, if the threshold is dependent on the number of the virtual machine.

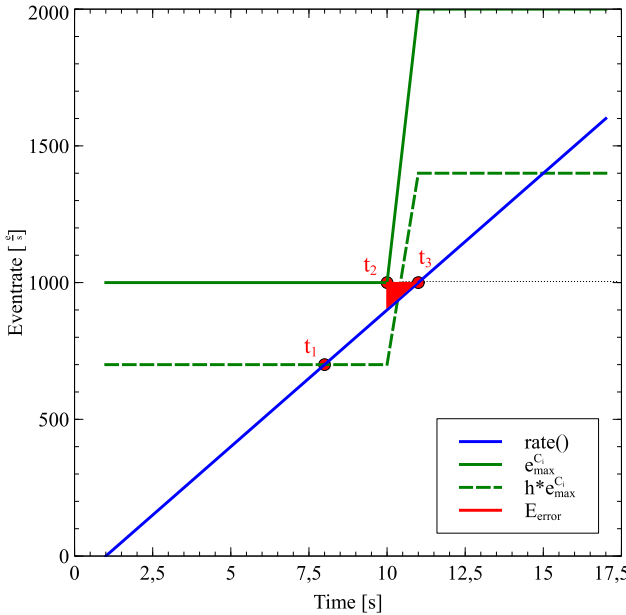


Figure 6.6: Test b:Threshold = 0,7

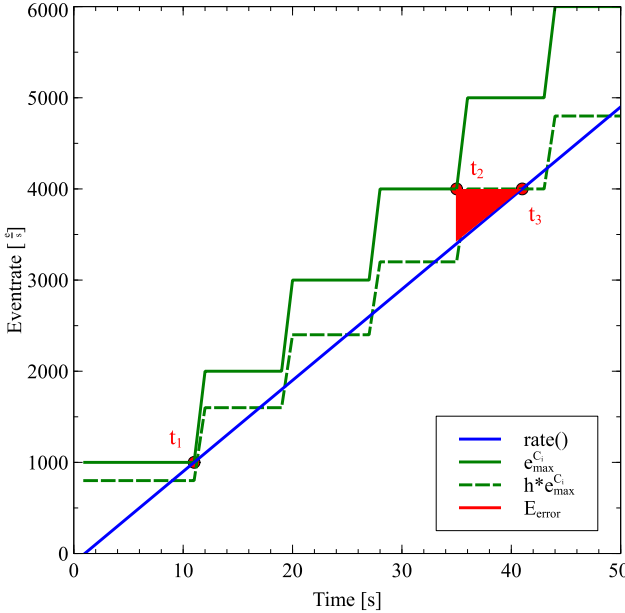


Figure 6.7: Long-term observation

Summary and Future Work

A challenge in CEP is to cope with the possibly strongly changing number of events be processed. The more events occur, the more events have to be processed and the more computing power is needed. There are times when more computing power is required and times when less computer power is required. Hence, the number of the required resources changes. The idea is to use Cloud Computing to dynamically allocate resources. In this work a new approach has been introduced to handle peaks in the workload by using a Cloud environment. When the workload increases as less as possible events are stored in a queue. Thus, as few as possible events are processed in delay.

Therefore, a system model was developed to deploy a rule network in a Cloud environment. The core idea is to run every rule on a single virtual machine and allocate another virtual machine with the same rule if more events have to process. Virtual machines running the same rule belong to the same cluster. Furthermore, different components as virtual node, rule cluster and Cloud Provider has been characterized in order to define the maximum eventrate as well to describe the allocation and deallocation process. With the maximum eventrate the workload can be determine.

Each cluster has a resource manager to handle the resources for one rule. The resource manager ensures that each cluster has enough resources to process all incoming events without delay and tries to keep the number of resources as small as possible. In addition, it handles the equal distribution of the input events to the different nodes in a rule cluster.

A decision criterion was presented to decide, when the allocation and deallocation request has to be sent to the Cloud Provider. Our criterion defines a critical workload for each cluster and if the cluster reach its critical workload then the resource manager sends an allocation request. On the other side, if the workload decline below a specified threshold, a deallocation request is sent.

In the evaluation two different kinds of errors have been introduced. Either too less resources are available and events has to been stored in the queue or more resources are allocated than needed. Additionally, it has been presented two different error metrics t_{error} and E_{error} in order to assess the results. The tests showed that it is important to have prior knowledge

about the behavior of the number of the incoming events in to order to choose a suitable threshold, because the relation between the function *rate()* and the threshold has impact on the error. Moreover, the long-term observation showed that the size of the critical workload is larger according to the number of virtual machines. For particular application it would be better, if the threshold is dependent on the number of virtual machines. It is difficult to choose a suitable threshold and often no prior knowledge about the eventrate is available. Therefore, another possibility could be to use a self-learning approach to adapt the threshold according to the strength of the error. In detail, if it is recognized, that the error is high and lots of the events have to be stored then threshold should be change to a smaller value.

The suggested decision criterion is used on the basis of the maximum eventrate, therefore no access to the event queue is needed. Thus, it can be used for CEP Engines that offer no access to the event queue as for example Esper. Additionally, no information about the CPU workload is needed from the Cloud Provider, which can be also not available.

Every cluster has its own resource manager, with that the problem of a centralized load balancer does not arise. Every node has the same limited resources and has to process the same number of events due to equally dividing the event input stream. The resource manager can be calculating with the stored current number of the virtual machines the workload of the Cluster and no communication between the nodes and the resource manager has to be established.

The problem is that in this approach the precondition is used, that all events can processed independently from each other. Only with that the resource manager can split the input stream equally. In general, this precondition is not valid. Therefore, other approach to distribute the incoming event stream to the different nodes has to be developed. It should be also considered how events can be processed concurrently. This could offer a possibility to split the event stream to virtual machines.

Bibliography

- [BKS⁺99] G. Banavar, M. Kaplan, K. Shaw, R. Strom, D. Sturman, Wei Tao. Information Flow Based Event Distribution Middleware. In *Electronic Commerce and Web-based Applications/Middleware, 1999. Proceedings. 19th IEEE International Conference on Distributed Computing Systems Workshops on, title=Information flow based event distribution middleware*, pp. 114–121. 1999. (Zitiert auf Seite 27)
- [EB09] M. Eckert, F. Bry. 21 - Complex Event Processing. 2009. (Zitiert auf den Seiten 11 and 13)
- [ER11] M. El-Refaey. Virtual Machines Provisioning and Migration Services. In *Cloud Computing*, pp. 121–156. John Wiley & Sons, Inc, 2011. (Zitiert auf den Seiten 5, 18, and 19)
- [Esp08] EsperTech, 2008. URL <http://www.espertech.com/products/esper.php>. (Zitiert auf den Seiten 5 and 14)
- [Esp12] Esper Reference Documentation, 2012. URL http://esper.codehaus.org/esper-4.5.0/doc/reference/en/pdf/esper_reference.pdf. (Zitiert auf Seite 13)
- [HLS11] W. Hummer, P. Leitner, B. Satzger, S. Dustdar. Dynamic Migration of Processing Elements for Optimized Query Execution in Event-Based Systems. In R. Meersman, T. Dillon, P. Herrero, A. Kumar, M. Reichert, L. Qing, B.-C. Ooi, E. Damiani, D. Schmidt, J. White, M. Hauswirth, P. Hitzler, M. Mohania, editors, *On the Move to Meaningful Internet Systems: OTM 2011*, volume 7045 of *Lecture Notes in Computer Science*, pp. 451–468. Springer Berlin / Heidelberg, 2011. (Zitiert auf den Seiten 5, 22, and 23)
- [KKP11] W. Kleiminger, E. Kalyvianaki, P. Pietzuch. Balancing Load in Stream Processing with the Cloud. In *Data Engineering Workshops (ICDEW), 2011 IEEE 27th International Conference on, title=Balancing load in stream processing with the cloud*, pp. 16–21. 2011. (Zitiert auf den Seiten 5 and 22)
- [LRE09] G. T. Lakshmanan, Y. G. Rabinovich, O. Etzion. A stratified approach for supporting high throughput event processing applications. In *Proceedings of the*

- Third ACM International Conference on Distributed Event-Based Systems, DEBS '09*, pp. 5:1–5:12. ACM, 2009. (Zitiert auf Seite 27)
- [LS08] D. Luckham, R. Schulte. Event Processing Glossary - Version 1.1. 2008. (Zitiert auf den Seiten 7, 11, and 13)
- [MFP] G. Mühl, L. Fiege, Pietzuch. *20 - Distributed Event-Based Systems*. (Zitiert auf den Seiten 5 and 12)
- [MG11] P. Mell, T. Grance. The NIST Definition of Cloud Computing. 2011. (Zitiert auf den Seiten 16 and 17)
- [MHM⁺03] Mitch Cherniack, Hari Balakrishnan, Magdalena Balazinska, Donald Carney, Ugur Cetintemel, Ying Xing, Stan Zdonik. Scalable Distributed Stream Processing. In *CIDR 2003 - First Biennial Conference on Innovative Data Systems Research*. 2003. (Zitiert auf den Seiten 5 and 21)
- [RLTB10] M. Randles, D. Lamb, A. Taleb-Bendiab. A Comparative Study into Distributed Load Balancing Algorithms for Cloud Computing. In *2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops*, pp. 551–556. IEEE, 2010. (Zitiert auf Seite 22)
- [Ste] Stefan Grohe Christoph Schlameuß Ralf Sommer. Performancevergleich von CEP-Engines. (Zitiert auf Seite 27)
- [TÇZ⁺03] N. Tatbul, U. Çetintemel, S. Zdonik, M. Cherniack, M. Stonebraker. Load shedding in a data stream manager. In *Proceedings of the 29th international conference on Very large data bases - Volume 29, VLDB '03*, pp. 309–320. VLDB Endowment, 2003. (Zitiert auf Seite 21)
- [VBB11] W. Voorsluys, J. Broberg, R. Buyya. Introduction to Cloud Computing. In *Cloud Computing*, pp. 1–41. John Wiley & Sons, Inc, 2011. (Zitiert auf den Seiten 5, 9, 16, 17, and 18)

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Eva Hoos)