KTH Electrical Engineering

Distributed Monitoring and Resource Management for Large Cloud Environments

FETAHI ZEBENIGUS WUHIB

Doctoral Thesis
Stockholm, Sweden, December 2010

TRITA-EE 2010:051
ISSN 1653-5146
ISBN 978-91-7415-794-9 KTH, School of Electrical Engineering

Akademisk avhandling som med tillstånd av Kungl Tekniska högskolan framlägges till offentlig granskning för avläggande av teknologie doktorsexamen fredagen den 10 december 2010 i sal Q2, KTH, Stockholm.

Tryck: Universitetsservice US AB.

Abstract

Over the last decade, the number, size and complexity of large-scale networked systems has been growing fast, and this trend is expected to accelerate. The best known example of a large-scale networked system is probably the Internet, while large datacenters for cloud services are the most recent ones. In such environments, a key challenge is to develop scalable and adaptive technologies for management functions. This thesis addresses the challenge by engineering several protocols for distributed monitoring and resource management that are suitable for large-scale networked systems. First, we present G-GAP, a gossip-based protocol we developed for continuous monitoring of aggregates that are computed from device variables. We prove the robustness of this protocol to node failures and validate, through simulations, that its estimation accuracy does not change with increasing size of the monitored system under certain conditions. Second, we present TCA-GAP, a tree-based protocol, and TG-GAP, a gossip-based protocol for the purpose of monitoring threshold crossings of aggregates. For both protocols, we prove correctness properties and demonstrate, again through simulations, that both protocols are efficient, by showing that their overhead is at least two orders of magnitude smaller than that of a naïve approach, for cases where the monitored aggregate is sufficiently far from the threshold. Third, we present a gossip-based protocol for resource management in cloud environments. The protocol allocates CPU and memory resources to applications that are hosted by the cloud. We prove that the resource allocation computed by the protocol converges exponentially fast to an optimal allocation, for cases where sufficient memory is available. Through simulations, we show that the quality of the resource allocation approaches that of an ideal system when the total memory demand decreases significantly below the memory capacity of the entire system. In addition, we validate that the quality of the allocation does not change with increasing the number of hosted applications and machines, for the case where both metrics are scaled proportionally. Finally, we compare two approaches (tree-based and gossip-based) to engineering protocols for distributed management, for the case of real-time monitoring. Results of our simulation studies indicate that, regardless of the system size and failure rates in the monitored system, gossip protocols incur a significantly larger overhead than tree-based protocols for achieving the same monitoring quality (e.g., estimation accuracy or detection delay).

Acknowledgements

First off, I would like to express my deepest gratitude to my advisor, Prof. Rolf Stadler, for accepting me into his group and for his continuous support and unwavering patience during the course of this thesis. It has been an honor and a privilege to have worked with him. I would also like to thank Dr. Mads Dam from CSC/KTH, for being the second advisor to the large part of this thesis.

Several people have contributed to the results presented in this work. In this regard, I would like to thank Dr. Alex Clemm from Cisco Systems and Dr. Mike Sprietzer from IBM Research, for suggesting many of the problems addressed in this thesis, and for their comments and contributions that played a critical role in shaping the work that resulted. I would also like to thank Prof. Danny Raz from The Technion for taking his time to be the opponent of both my licentiate and Ph.D. defenses. His comments on my licentiate thesis have been quite useful for this thesis.

I would like to thank everyone at LCN, for the excellent research atmosphere and for the helpful comments I got during the course of my work. I would also like to thank the people at IBM Research for hosting me during my internship: my managers Dr. Giovanni Pacifici and Dr. Malgorzata Steinder, my supervisor Dr. Mike Spreitzer, as well as Dr. Ian Whalley and Dr. Claris Castillo who made sure that I never felt alone during my stay there.

Who I am today is greatly owed to my parents, my mother Almaz Hussien and my late father Zebenigus Wuhib. Their primary aim in life has been to ensure that their children have education that guarantees a secure future. Though they struggled at times to make ends meet, they were never late to pay the school fees for the private schools we attended. I hope they feel that their struggles have been rewarded by who we are today. In particular, I know that my father would have loved to see this date. Ababa, here is my Ph.D. thesis, as I promised you ten years ago on my graduation from AAU.

Outside of research, I am very thankful for the wonderful people that surround me. Thank you, my wife Tigi and daughter Eeboo, for showing me the meaning of life. Because of you, these last five years of my life have simply been the best. Thank you, Tigist and Eyosias, for providing my family with a home away from home. Because of you, we have never missed our homes, especially on Ethiopian holidays. Thank you, Barbro and Roger Brandt, my wife and I are proud to call you our 'Swedish parents'. Thank you, Mehila and Darik, for taking care of business back at home. I always felt that the fix to any problem was a phone call away to you guys. Thank you Eskindir, for proofreading all my theses, starting from our B.Sc. thesis we worked on together. Now, anyone who finds a typo in them knows who is to blame.

Fetahi Zebenigus Wuhib
Stockholm, December 2010.

Table of Contents

Chapter 1

Introduction

1.1 Background and Motivation

The size and complexity of large-scale networked systems (LNSs) has grown fast over the last decade, and this trend is expected to accelerate further. The Internet, for instance, is now expected to connect over five billion devices, a number that will likely grow to some 22 billion by 2020[35]. Using the internet as a basis, other LNSs are emerging. Examples include peer-to-peer file sharing networks, which are expected to be the second largest source of traffic in the coming five years, according to a Cisco study[14]. Another example is the Skype network, the largest peer-to-peer VoIP network, which currently counts 124 million unique users each month[67].

The latest manifestations of LNSs are datacenters that provide *cloud computing* services. Some datacenters that are currently built will contain hundreds of thousands of servers[82, 52, 62]. Cloud services follow a paradigm where resource demanding computation is moved from end-user devices (e.g., desktops, laptops and smart phones) to datacenters that are owned and operated by a third party. This paradigm is attractive for users of the service, as it gives them dynamic access to resources and satisfies their changing needs. It enables cloud service providers to exploit the economy of scale, as the cost per machine falls for increasingly large datacenters.

The LNSs outlined above cannot be effectively and efficiently managed using traditional management systems that follow a centralized management paradigm. This is because the number of states and events that need to be monitored, as well as the number of control actions that need to be determined and executed become too large to handle. As a result, there is an urgent need for new management solutions that scale with the size of the managed system.

We have identified three properties of a management system that is suitable for LNSs. The first property is architectural and suggests using a decentralized management architecture where basic monitoring and control functions are pushed into the managed system and thus enable scalable operation and short reaction

times[71]. The research challenge here is the engineering of management protocols that can run in such a distributed setting. The second property relates to how the managed system is abstracted and calls for functions that can efficiently estimate the global state and how it evolves over time. The challenge here is to develop accurate and efficient protocols for the monitoring of global state variables that are computed from local state variables. The third property relates to how state variables are accessed for monitoring purposes and advocates a push approach, in contrast to the pull approach that is ubiquitous in traditional management systems. It states that the managed system itself should identify and push updates and alerts to the management system, which allows it to quickly take appropriate actions.

The research presented in this thesis focuses on engineering a small set of management protocols that perform key functions in a LNS and exhibit the three properties given above. This work is an extension of the author's licentiate thesis[88]. (A licentiate at KTH is an intermediate degree between an M.Sc. and a Ph.D. degree.)

1.2 The Problem

The functions of a management system are often described in terms of the five functional areas – fault management, configuration management, accounting management, performance management and security management – which are abbreviated as FCAPS[36]. A second categorization of management functions is distinguishing between *monitoring* functions and *control* functions. In this thesis, we contribute towards monitoring, as this is a key building block for all FCAPS functions. Second, we contribute towards control, by studying resource allocation, which is an essential aspect of performance management.

Monitoring in Large Cloud Environments

Monitoring is the process of acquiring state information from a managed system. In traditional network and systems management, monitoring is performed on a per-device basis, whereby a management station periodically polls devices in its domain for the values of local variables, such as device counters, performance parameters or identifiers of flow that currently traverse the device. These variables are then processed on the management station to compute an estimate of a network-wide state, which is analyzed and acted upon by management programs. SNMP is probably the best known protocol that supports this type of monitoring, which follows the *pull* approach[71]. An alternative to the pull approach, which is based on polling, is the *push* approach, whereby network elements send values of local variables to the management station whenever changes to those values occur. With the emergence of large and dynamic networked systems, the push approach is gaining importance, because this approach enables the management system to continuously follow the evolution of the network state.

We present the monitoring problem by modeling the managed system as a dynamic set of nodes $V(t)$ that represent the devices. Over time, nodes may join

or leave the system, or they may fail. To each node $i \in V(t)$ is associated a local variable $x_i(t) \geq 0$. Central to our work is the monitoring of a global variable $A(t) = \Pi_{i \in V(t)}(\{x_i(t)\})$, which is computed from local variables $x_i(t)$ using an aggregation function Π. Examples of such aggregation functions include SUM, extremal functions, histogram, as well as statistical means and moments (see Chapter 6). More generally, we consider in our work aggregation functions that are both commutative and associative.

In this setting, three classes of protocols can be studied and engineered.

1. **Polling of aggregates:** The problem is to develop a protocol for computing a *snapshot* (or alternatively, estimating the value) of a global variable $A(t) = \Pi_{i \in V(t)}(\{x_i(t)\})$ at a time $t = t_0$.

2. **Continuous monitoring of aggregates:** The problem is to develop a protocol for *continuously* estimating $A(t) = \Pi_{i \in V(t)}(\{x_i(t)\})$ (see Figure 1.1a).

3. **Monitoring threshold crossings of aggregates:** The problem is to develop a protocol that raises an alert when an aggregate $A(t) = \Pi_{i \in V(t)}(\{x_i(t)\})$ crosses from below a given global threshold T_g^+ and that clears the alert when the aggregate crosses a second, lower threshold T_g^- from above (see Figure 1.1b).

In order to effectively and efficiently execute in large, dynamic networked systems, the design goals for these three classes of protocols are set out as follows:

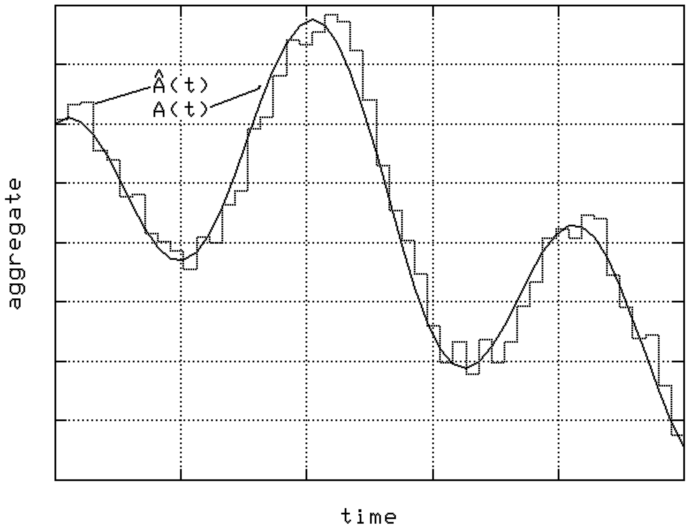
Efficiency: The protocol overhead, measured, e.g., as the number of messages per time unit that are exchanged among system components, must be as small as possible.

Quality: The protocol must achieve a high quality in state estimation or event detection, for a given protocol overhead. This means that the protocol must exhibit a small error in estimating an aggregate, or a small delay when detecting a threshold crossing. It must also exhibit a low probability of false positives and false negatives when detecting such a crossing.

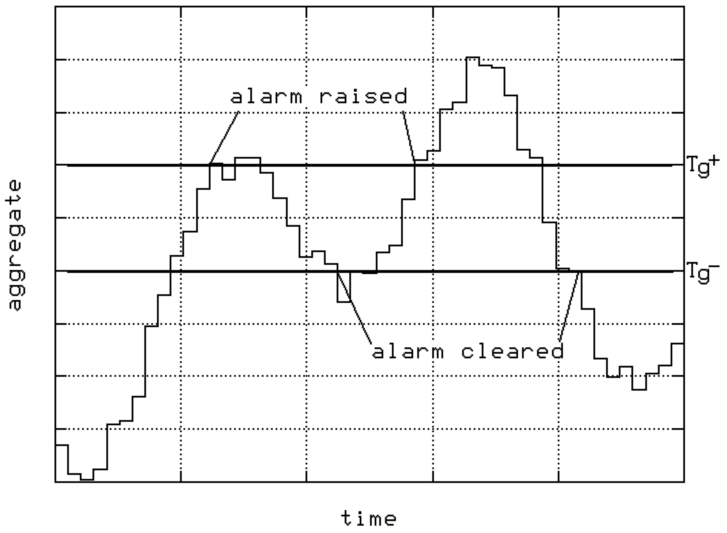
Scalability: Both of the above metrics, efficiency and quality, must scale with the system size. Specifically, we require that the quality of the protocol, for a given overhead, degrades sub-linearly with increasing system size; similarly, the overhead must increase sub-linearly with the system size, for a given quality goal.

Robustness: The protocol must dynamically adapt, in order to ensure continuous operation after failures or node addition and removal.

Controllability: The protocol must allow for controlling the trade-off between the quality of the monitoring activity and the protocol overhead. This capability must be dynamic, in the sense that one must be able to turn at runtime the 'management knobs' that implement this trade-off.



(a) Continuous monitoring of aggregates. $A(t)$ is the real value while $\hat{A}(t)$ is the computed estimate.



(b) Monitoring threshold crossings of aggregates. T_g^+ and T_g^- are upper and lower thresholds, respectively.

Figure 1.1: Continuous monitoring of global variables.

This thesis addresses the problem of continuous monitoring of aggregates and the problem of monitoring threshold crossings of aggregates for the design goals given above. It does not further investigate the problem of polling aggregates, as many solutions are available in the recent literature (cf. [47, 42, 38, 6, 59] and those listed in [69]).

Resource Management in Large Cloud Environments

In the context of large-scale distributed computing, a key problem in resource management is that of mapping a set of applications onto a system of machines that execute those applications and, for each such machine, assigning local resources for those applications that run on it. The quality of the allocation process is often measured through a utility function, which is an aggregation function computed from local state variables. An optimal allocation maximizes such a system utility. The machine resources that are allocated to applications include CPU, memory, storage, network bandwidth, access to special hardware/software, etc. The resource demand of an application can change over time. In response to such changes, the resource allocation process needs to be repeated many times over; in other words, it has to be dynamic, in order to ensure that the system utility is maximized at all times.

Optimal resource allocation in the sense of utility maximization is often computationally expensive. In the context of *grid computing*, for instance, the problem of scheduling jobs onto machines such that the total execution time is minimized can be formulated as the *minimum makespan scheduling problem*, which is known to be NP-hard[29]. Second, in the context of cloud computing, the problem of placing applications onto machines is often modeled as a variant of the *knapsack problem*, which is also known to be NP-hard[74].

In this work, we model the managed system as a dynamic set of nodes that represents the machines of a cloud environment. Over time, nodes may join or leave the system, or they may fail. Each node has a specific CPU capacity and a memory capacity. This system executes a number of applications, each of which is composed of a set of modules. The external load on these applications (and, implicitly, on the modules themselves) varies over time. We solve the problem of finding a mapping of applications (or, more precisely, the modules) onto nodes and a local resource allocation policy, such that a given system utility is maximized at all times. We are specifically interested in a system utility that achieves max-min fairness[13, 18] for CPU resources under memory constraints. Max-min fairness maximizes the minimum utility of any application in the system.

We identify the design goals for a protocol that effectively and efficiently performs resource allocation in large, dynamic systems as follows:

Efficiency: The protocol overhead, measured, e.g., as the number of messages per time unit that are processed by a node, must be as small as possible.

Quality: The protocol must achieve, to the maximum extent possible, max-min fairness for computational resources under memory constraints.

Adaptability: The resource allocation process must dynamically and efficiently adapt to changes in external load and changes in system resources.

Scalability: The resource allocation process must be scalable both in the number of nodes and the number of applications. This means that the resources consumed per node in order to achieve a given performance objective must increase sublinearly both with the number of nodes and with the number of applications.

Controllability: The protocol must allow for controlling the trade-off between the quality of resource allocation and the protocol overhead. This capability must be dynamic, in the sense that one must be able to turn at runtime the 'management knobs' that implement this trade-off.

We study the problem of resource allocation in the context of a cloud service provider that owns and administrates the physical infrastructure of a datacenter. Using this infrastructure, the service provider offers application hosting services to its clients. The clients own these applications and use them to provide services to end users through the public Internet.

This thesis contributes towards a resource management architecture for a cloud environment and it proposes a resource allocation protocol that supports the above design goals in the context of such an architecture.

1.3 The Approach

Design Principles

In the context of large-scale networked systems, the goals set out above for monitoring and resource allocation protocols are difficult, if not impossible, to achieve using traditional, centralized management architectures. In centralized management systems, the computational complexity of management tasks resides almost entirely in management stations, as opposed to in elements of the networked system. In order to estimate a global variable, for instance, a management station collects individual values from devices in the managed system and then performs the aggregation process. Since the load on the management station and the time needed to execute the management task increases (at least) linearly with the system size, the overhead and the delay associated with such a task can become prohibitively large when the system reaches a certain size.

To address the outlined problem of scalability, we rely on the use of distributed and adaptive algorithms for monitoring and resource allocation. (While we use here the terms 'distributed algorithm' and 'protocol' with the same semantics, we mostly apply the term protocol in the remainder of the thesis.) A first property that the protocols we propose in this work share is that all elements involved in a specific management operation execute the same functions. A second property is

that each element only maintains partial knowledge of the networked system and thus interacts only with a (small) subset of all elements in the system.

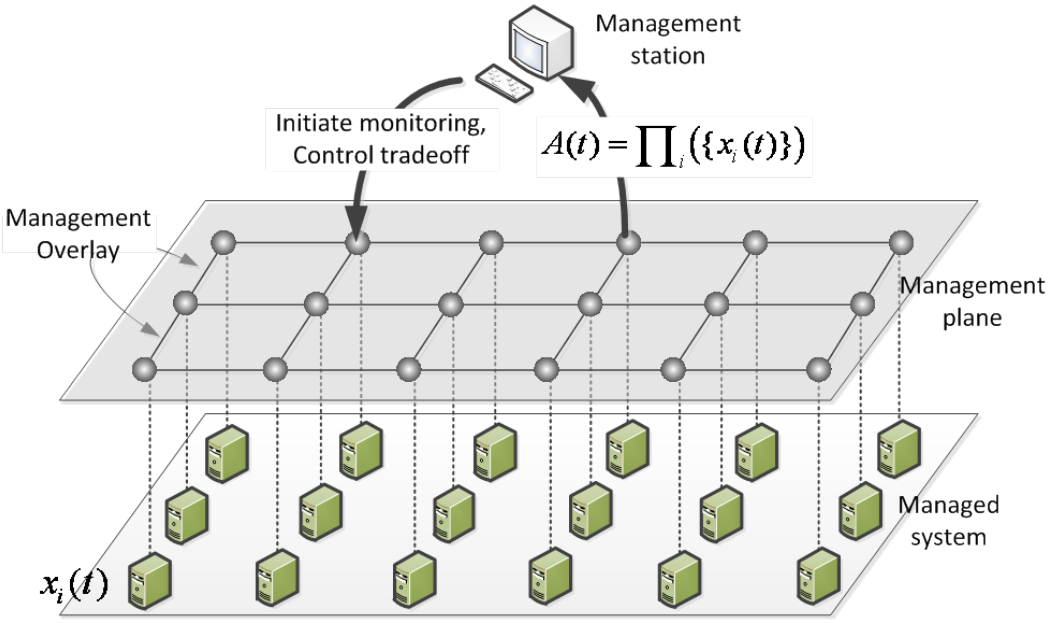


Figure 1.2: Our architecture for distributed monitoring. Our distributed algorithms run in the management plane.

Figure 1.2 outlines the architectural framework we use for distributed monitoring (cf. [57, 97]). In this framework, we associate a management process that has access to the local variables with each device of the managed system. The management processes self-organize into a global management plane that provides the desired monitoring functionality to management applications. A protocol (such as [85, 80, 37, 39]) creates and maintains a management overlay in this plane, which defines a network graph. The monitoring protocols which we devise execute in the management plane and run on this network graph. The topology of the network graph influences the performance of the monitoring protocols. The management station initiates monitoring tasks in the management plane and controls the tradeoff between the quality of the monitoring process and the overhead incurred by the protocol in the management plane. Typically, results from the monitoring tasks are pushed from the management plane to the management station in a continuous fashion.

In analogy to Figure 1.2, Figure 1.3 outlines our architectural framework for resource management. Similarly, we associate a management process that has access to the local variables with each machine of the managed system. The management processes self-organize into a global management plane that performs the task of resource allocation. The resource management protocols execute in the management plane. They compute the local resource allocation policies for each machine and

use the network graph for peer selection. The management plane is realized as part of the cloud middleware. The management station is responsible for addition and removal of applications, setting performance parameters and controlling the tradeoff between the quality of resource allocation and the overhead incurred by the protocol in the management plane.

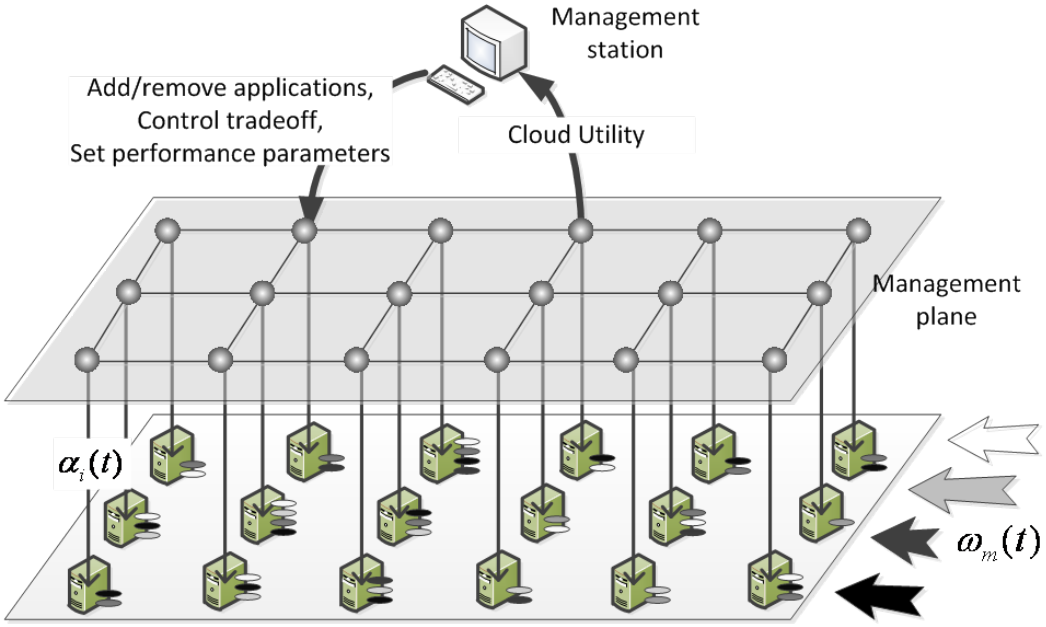


Figure 1.3: Our architecture for distributed resource management. The resource allocation protocol runs in the management plane. $\alpha_i(t)$ controls the (local) load on machine i while $\omega_m(t)$ is the (global) load for application m.

The protocols considered in this work can be classified into either tree-based or gossip-based. *Tree-based protocols* create and maintain a spanning tree in the management plane whose nodes are the management processes. Computational tasks, such as incremental aggregation, are preformed using this tree.

Gossip protocols are round-based protocols where, in each round, a node selects a subset of other nodes to interact with. Node selection is often probabilistic. As gossip protocols do not maintain a global data structure, such as a tree, they tend to be simpler than tree-based protocols. However, they tend to be less efficient in the sense that the overhead of a gossip protocol is often larger than that of a tree-based protocol for the same management task. An essential difference between the two classes of protocols is that tree-based protocols generally provide deterministic results while gossip protocols are often probabilistic, with their state variables converging towards increasingly accurate results over time.

The protocols proposed in this thesis are engineered from simpler protocols known from the literature. For the purpose of continuous monitoring of global

aggregates and threshold detection, we rely on basic protocols for distributed aggregation. In the case of gossip-based aggregation, we use *Push-Synopses*[42] and, in the case of tree-based aggregation, we use GAP[16]. For the purpose of distributed resource allocation, we build upon the gossip-based aggregation protocol by Jelasity et al.[38].

Evaluation

Our evaluation methodology includes three complementary parts. The first part uses theoretical analysis of key protocol aspects, including correctness and convergence. While analysis provides general insight into fundamental properties of a protocol, simulation studies are needed to fully understand the protocol behavior in most realistic scenarios. Such studies enable us to assess the protocols for specific parameter ranges of interest, with respect to our design goals – efficiency, quality, scalability, robustness and controllability. Since simulation is a flexible and powerful tool, a large part of the evaluation for this work is based on simulation results. As a downside, we need to develop our own simulation environment in order to exploit the full potential of simulation. The third method for protocol evaluation consists of implementation and experimentation on a testbed, which allows us to demonstrate the feasibility of realizing the intended management function with a specific technology.

1.4 The Contribution of this Thesis

We believe our thesis makes significant contributions towards engineering efficient and effective functions for monitoring and resource management in large-scale networked systems. The tangible results of this work can be summarized as follows.

I. Continuous Monitoring of Global Aggregates

We have developed a gossip-based protocol that we call G-GAP[93, 87] for continuous monitoring of global aggregates in large-scale network systems. G-GAP implements a novel approach to handling crash failures, which does not require restarting the protocol, as advocated in [38]. The protocol is robust against non-contiguous node failures, which are failures whereby neighboring nodes do not fail within two protocol rounds. We analytically prove the correctness G-GAP by showing that the protocol invariants are reinstated after non-contiguous failures.

Our extensive simulation studies show that G-GAP fulfills the design goals given in Section 1.2, for the parameter ranges studied. Specifically, we show that (1) an estimation error of less than 5% is achieved for traces of local variables that are based on available traffic measurements; (2) the estimation accuracy of the protocol, for a given overhead, stays the same across the range of network sizes tested; (3) the tradeoff between estimation accuracy and the protocol overhead can be effectively controlled.

II. Monitoring of Threshold Crossings of Aggregates

We developed two protocols for monitoring threshold crossings of aggregates: a tree-based protocol we call TCA-GAP and a gossip-based protocol we call TG-GAP[91, 97, 89, 90, 86]. The unique property of these protocols is that they exhibit low or no overhead when the monitored aggregate is far from the threshold.

Regarding TCA-GAP, we analytically prove correctness, i.e., correct detection of all threshold crossings, under the assumption that communication and processing delays can be neglected. The protocol is evaluated regarding the design goals through simulation studies using traces based on traffic measurements, and the results show that these goals are met. Specifically, (1) the protocol is efficient in the sense that it exhibits no overhead when the aggregate is sufficiently far from the monitored threshold, and (2) the protocol is scalable in the sense that its overhead does not increase with growing system size, while the delay associated with detecting threshold crossings tends to grow with the logarithm of the system size. Finally, we have implemented TCA-GAP on our laboratory testbed and evaluated its performance for a flow monitoring application. The testbed measurements confirmed conclusions about the protocol behavior we have been drawing from the simulation studies.

With respect to TG-GAP, we give a proof of correctness under the assumption that the local variables do not change. Through simulations, we show that TG-GAP fulfills our design goals given before. Key findings from simulations include that (1) TG-GAP is efficient in the sense that, within the parameter ranges studied, its overhead is often two orders of magnitude smaller than that of a naïve, straightforward approach to threshold detection (depending on the distance of the aggregate from the threshold), and (2) the protocol is scalable in the sense that its overhead and detection delay do not increase with the system size.

III. Resource Management for Cloud Environments

We developed a gossip-based protocol for resource allocation in large-scale cloud environments. The protocol performs a key function within a distributed middleware architecture for large clouds[98, 99].

The protocol implements a distributed scheme that allocates cloud resources to a set of applications that have time-dependent CPU demands and time-independent memory demands, and it dynamically maximizes a global cloud utility function. We analytically prove that the protocol produces a distributed allocation that converges exponentially fast to an optimal allocation when memory constraints can be neglected.

As in the above cases, the protocol is shown to meet its design goals through simulation. Key findings of the simulations include that (1) the protocol produces an allocation close to optimal when the aggregated memory demand is sufficiently smaller than the memory available in the cloud, and (2) for a given overhead, the quality of the allocation does not change with the number of applications and the

number of machines, for cases where the system size increases proportionally with the number of applications.

IV. Performance Comparison of Tree-based vs. Gossip-based Protocols

An insight we gained through our work is that important functions in distributed management can be provided both through tree-based and gossip-based approaches. We have shown this for the problem of real-time monitoring of global variables. (We believe that this fact extends to resource allocation as well.) Therefore, it is important to compare one approach against another with respect to the design goals we set out for distributed management operations (see Section 1.2). We have concluded that a theoretical comparison is beyond the scope of this work. However, we have conducted extensive simulation studies in which we compare quality metrics of representative protocols from both approaches for comparable overhead[87, 86]. A key finding from this work is that tree-based protocols are better suited than gossip-based protocols for monitoring in the types of scenarios we investigated. For example, regarding continuous monitoring of aggregates, we observe that the error in estimating an aggregate made by the gossip-based protocol is generally an order of magnitude larger than that made by the tree-based protocol. We made a similar finding with regards to threshold detection: for a comparable overhead, the detection delay for the tree-based protocol is clearly smaller than that for the gossip-based protocol, within the parameter ranges investigated.

Publications

The results of this research have been documented in ten peer-reviewed publications. Three of them have been published in journals (IEEE TNSM and Computer Networks), one in a magazine (IEEE Communications) and six in conferences of the network management research community (IEEE IM, IEEE DSOM, IEEE CNSM, IEEE E2EMON, ACM LADIS). These publications are listed in Chapter 5.

Chapter 2

Related Research

The work presented in this thesis focuses on three research areas: monitoring of system-wide aggregates, monitoring of threshold crossings of system-wide aggregates and resource management in cloud environments.

2.1 Distributed Monitoring of Global Aggregates

The primary reason for research in the distributed monitoring of aggregates is to achieve scalability in the sense that key performance metrics of the aggregation protocol degrade sub-linearly with the system size. Existing work in this area can be categorized according to how aggregates are computed and whether the protocol provides a snapshot or a continuous estimate of the aggregate.

Distributed Approaches to Computing an Aggregate

Several protocols for distributed computation of aggregates have been proposed in the literature (e.g., [72, 4, 6, 38, 59, 54, 16, 64]). These protocols typically execute on a network graph that interconnects the nodes of the system. Each node only has partial knowledge of the system, and interacts only with its neighbors on the graph. Based on the interaction patterns of these protocols, the work in [47] classifies them into four classes: flooding-based, random walk-based, tree-based and gossip-based.

Flooding-based protocols (e.g., [72, 4]) are protocols where a node whose state changes initiates a flooding of its new state within the network by sending its state information to all its neighbors, which is then forwarded by the receiving nodes until the state reaches all nodes. Consequently, all nodes receive updates of the states of all other nodes in the network. From this information, a node can compute an estimate of the global aggregate. This approach is not scalable, since the load on any node increases linearly with the system size, for a given update rate on a node. In [4], the authors use order and duplicate insensitive representations of nodes' states in order to reduce the size of the state of the protocol, which comes at the expense of increased protocol error.

Random-walk-based protocols (e.g., [4, 6]) are protocols where the state of a node is propagated using a random walk traversal algorithm. A node that starts the walk sends its state information to a random neighbor. The receiver updates its state and forwards the new state to another neighbor selected at random. This way, the state of a node reaches all other nodes in the system. To speed up the spread of state information, multiple random walks may be run at the same time. A problem inherent to random walk-based aggregation protocols is the problem of scalable state representation. Specifically, the problem is that of representing partial aggregate information on nodes in a compact way that avoids double counting. The common solution is to use order and duplicate insensitive techniques (cf. [27]) for state representation and computation of partial aggregates. A major drawback of this approach is that estimation error of computed aggregates can be high (e.g., typically larger than 20% in [4]).

We do not consider the above classes of protocols for our work due to the scalability and accuracy problems inherent to the way the aggregates are computed.

Tree-based protocols (cf. [59, 54, 16]) are those protocols where nodes organize themselves into a spanning tree. On this tree, the global aggregate is computed incrementally, by leaf nodes sending updates of their local variables to their respective parents, and by each parent computing its partial aggregate from updates it receives from its children. The aggregate computed at the root node becomes the global aggregate. Tree-based computation of aggregates requires that the tree structure be maintained after node failures or additions. A comprehensive review of tree-based aggregation protocols is presented in [69].

Gossip protocols (also known as epidemic algorithms) are round-based protocols characterized by asynchronous and often randomized communication among participating nodes. In gossip-based aggregation protocols, each node holds an estimate of the aggregate that typically converges exponentially fast to the true aggregate (cf. [42, 38, 64]). Gossip protocols are known to be robust against node failures when applied to information dissemination[19]. However, when applied to distributed computation of aggregates, a protocol invariant, also referred to as "mass"[42], needs to be maintained under node failures, which is non-trivial. Several examples of gossip-based aggregation protocols from the literature are discussed in [87].

Polling vs. Continuous Monitoring

We can further distinguish between approaches that compute a snapshot of the system state (also known as 1-time queries), which corresponds to polling in the context of network management, and approaches that provide a continuous estimation of an aggregate, which corresponds to continuous monitoring.

Polling protocols have been extensively studied in the literature (cf. [47, 42, 38, 6]; see also the survey in [69]). In this work, we do not focus on polling protocols as they are unsuitable for maintaining state information in real-time. Note that continuous monitoring can be approximated through periodic polling or so-called

N-time queries (e.g., see [38, 59]), at the risk of missing changes to the aggregate during the polling intervals.

Protocols for continuous monitoring of aggregates have also been proposed in the literature (e.g., [16, 70, 4, 77, 65, 93, 87]). The common property of these protocols is that they use *incremental aggregation* techniques. In incremental aggregation, a new estimate of the aggregate is computed from an existing estimate and information about the changes in local variables. Our work focuses on continuous monitoring (as opposed to polling) since it enables management systems to continuously follow the evolution of the network state.

2.2 Detecting Threshold Crossings of Global Aggregates

A naïve solution for detecting threshold crossings of aggregates is to use a protocol for continuous monitoring of aggregates and evaluate the threshold conditions each time the aggregate changes. Although straightforward, this solution is not efficient and may become infeasible if the network is large or a significant number of threshold crossings of different aggregates need to be monitored.

Recently, several results in detecting threshold crossings of aggregates have been published (cf. [20, 45, 33, 76, 89, 11, 86, 48]). The common approach in achieving efficiency has been reducing protocol overhead, compared to the naïve solution, when the aggregate is far from the threshold. The approach includes the concept of local thresholds on the nodes that run the protocol. In general, the local thresholds define conditions under which nodes send updates of their local states, thereby reducing the protocol overhead. The problem then becomes that of setting local thresholds in such a way that the overhead is significantly reduced, while still achieving correct detection of threshold crossings.

We classify recent research according to the level of decentralization of the approaches and the type of aggregation functions supported by the protocols. A more detailed presentation of the works given here can be found in [89].

Weakly distributed protocols (e.g., [20, 45, 33, 76]) assume a coordinator that computes the local thresholds for all nodes. This coordinator also computes the aggregate and detects the crossings of the global threshold. A common drawback of having a coordinator is that the load on the coordinator generally increases linearly with the system size, for a given update rate of local variables. As a result, we do not consider this class of protocols in this work.

In *strongly distributed* protocols (e.g., [89, 11, 76]), the aggregate and local thresholds are computed in a distributed manner. For instance, in our tree-based protocol[89], the parent computes the thresholds for its children. In [11], a node computes a threshold for its subtree and in [76, 86], each node computes its own threshold.

Based on the type of aggregation function supported by the protocol, we classify protocols into those that support (one or more) *simple* aggregation functions (e.g.,

[20, 45, 89, 11, 86, 48]) and those that support *complex* aggregation functions (e.g., [33, 76]). Examples for simple aggregation functions include COUNT, SUM, AVERAGE and MAX while examples for complex aggregation functions include computing principal components[33]. The methods for computing the aggregate and the local thresholds for such functions are well understood. However, the theory for computing, in a distributed fashion, aggregates and local thresholds for functions other than the ones listed above is not well developed today. In this regard, [33, 76] make relevant contributions.

2.3 Resource Management for Cloud Environments

Our work on resource management for cloud environment overlaps with the following research areas.

Resource Management in Grid Computing

Grid computing is a computing paradigm that was born out of the need for an abstraction that allows using a wide range of heterogeneous, loosely coupled resources in a consistent manner[28]. The computational resources are typically owned by different organizations and fall under different administrative domains.

Both grid computing and cloud computing aim at making scalable computational resources available to users that do not own and operate the associated physical infrastructure. However, there are important differences between the two. First, in cloud computing, there is a single cloud service provider that sells its services to a wide range of users. In contrast, the physical infrastructure providing grid services is owned and operated by multiple organizations and the services it provides are often restricted to users authorized by the organizations. Second, the types of applications targeted by grid computing are generally CPU intensive, non-interactive applications that are often run as batch jobs.

The key problem of resource management in grids is that of job scheduling with the goal of minimizing a cost (e.g., completion time), which in general is NP-hard[40]. A survey of existing grid resource management systems and job scheduling algorithms is available in the literature[40, 50, 46].

The problem of resource management in cloud computing is different in the sense that applications targeted by cloud computing are often interactive and as a result, their demand can vary highly over time. This means that the allocation of resources to applications needs to be recomputed frequently, compared to the case of grid computing.

Resource Management in Cloud Environments

A key problem of resource management in cloud environments centers around the problem of dynamic application placement with the goal of maximizing cloud utility.

A number of centralized (cf. [79, 25, 13]) and decentralized (cf. [3, 102, 58, 7]) solutions have been proposed in the literature to the problem of dynamic application placement. As centralized approaches, in general, suffer from scalability problems, we do not consider them further.

For the case where the resource demand of an application can be split over several machines, it is possible to design algorithms that give optimal solutions (cf. [58]). However, some resource demands, such as demand for memory, are typically indivisible. In such cases, the placement problem becomes NP-hard[29] and solutions are often heuristic (cf. [79, 25, 13, 3]).

Similar to ours, the work in [102] considers the problem of virtual machine placement under CPU and memory constraints. There, the authors use multi-criteria decision analysis to compute the placement in a decentralized manner. Unlike us, their solution assumes the existence of an oracle with global information regarding the machines and their remaining capacities, which limits the scalability of their approach.

Another decentralized solution is proposed in [3, 24] where the authors present a distributed middleware layer for application placement in datacenter environments. There, given a set of applications with relatively large demand, machines start or stop instances of the applications, such that a cluster utility (with the utility function different from the one considered in this work) is maximized. The design scales in the number of machines but, unlike our design, does not scale in the number of applications.

Distributed Load Balancing

Our protocol for resource allocation in cloud environments essentially reduces to a distributed load-balancing protocol when only CPU resources are considered. Such algorithms have been extensively studied for systems composed of homogeneous and heterogeneous machines, as well as for both divisible and indivisible demands. A particularly well-studied class of protocols are *diffusion algorithms* (e.g., [15, 22, 31, 32]) and *dimension exchange algorithms* (e.g., [100, 101, 5]). Convergence results for different network topologies and different norms (that measure the distance between the system state and the optimal state) have been reported, and it seems to us that the problem is well understood today. What makes the resource allocation problem considered in this work hard is the introduction of memory demands that can not be split across multiple machines.

Chapter 3

Summary of Original Work

The work presented in this thesis has led to ten peer-reviewed publications that are listed in Chapter 5. Five of these papers are included in the present thesis.

3.1 Robust Monitoring of Global Aggregates through Gossiping

We examine the use of gossip protocols for continuous monitoring of network-wide aggregates. Aggregates are computed from local management variables using functions, such as AVERAGE, MIN, MAX, or SUM. A particular challenge is to develop a gossip-based aggregation protocol that is robust against node failures. In this paper, we present G-GAP, a gossip protocol for continuous monitoring of aggregates, which is robust against discontiguous failures (i.e., under the constraint that neighboring nodes do not fail within a short period of each other). We formally prove this property, and we evaluate the protocol through simulation using real traces. The simulation results suggest that the design goals for this protocol have been met. For instance, the trade-off between estimation accuracy and protocol overhead can be controlled, and high estimation accuracy (below some 5% error in our measurements) is achieved by the protocol, even for large networks and frequent node failures. Further, we perform a comparative assessment of G-GAP against a tree-based aggregation protocol using simulation. Surprisingly, we find that the tree-based aggregation protocol consistently outperforms the gossip protocol for comparative overhead, both in terms of accuracy and robustness.
This paper appears in this thesis as Chapter 6. It is also published as:

> F. Wuhib, M. Dam, R. Stadler and A. Clemm, "Robust Monitoring of Network-Wide Aggregates Through Gossiping", **IEEE Transactions on Network and Service Management**, Vol.6, No. 2, pp.95-109, June 2009.

My contribution to the work has been the protocol design, implementation and evaluation. In addition to fruitful discussions on all aspects of this work, Mads Dam

from KTH helped with formalizing the protocol and proved the convergence properties. Alex Clemm from Cisco gave input to the protocol design and helped with application scenarios.

3.2 Service-Level Monitoring using Global Threshold Crossing Alerts

Service level agreements (SLAs) are at the core of the business relationship between service providers and their customers. Service level monitoring is necessary to validate and ensure that service levels are indeed adhered to. One key tool for this is threshold crossing alerts (TCAs). TCAs can notify a service provider that a certain parameter has exceeded a certain threshold value, directing attention to those areas where preventive action needs to be taken. This paper presents a protocol and an architecture that implements a new category of TCAs for parameters that need to be aggregated across a network, as opposed to parameters that can be observed from a single device - for example, "average utilization across all links in a network" as opposed to "utilization of a particular link". Although highly useful, such parameters are rarely subjected to SLAs today, due to the lack of effective monitoring technology. Our system fills this gap. It does so in a manner that is decentralized inside the network and does not rely on a more traditional centralized management architecture. We will focus on business application aspects, as well as report on test-bed results with respect to the robustness and accuracy properties of our system.
This paper appears in this thesis as Chapter 7. It is also published as:

> F. Wuhib, R. Stadler and A. Clemm, "Decentralized Service Level Monitoring using Network Threshold Crossing Alerts," **IEEE Communications Magazine**, Vol. 44, Issue 10, October 2006.

My contribution to this work has been the design, implementation and evaluation of the protocol. Alex Clemm originally suggested addressing the problem of threshold detection and gave helpful comments during the development of TCA-GAP.

3.3 Tree-based Detection of Global Threshold Crossings

The timely detection that a monitored variable has crossed a given threshold is a fundamental requirement for many network management applications. A challenge is the detection of threshold crossings of network-wide variables, which are computed from device counters across the network, using aggregation functions such as SUM, MAX and AVERAGE. This paper contains a detailed description and a comprehensive evaluation of TCA-GAP, a protocol for detecting threshold crossings of network-wide aggregates in a distributed way. Elements of its design

include tree-based incremental aggregation for estimating the value of aggregates, a local hysteresis mechanism to reduce overhead and dynamic recomputation of local thresholds to ensure correctness. The protocol is evaluated through extensive simulation using real traces in scenarios with network sizes up to 5232 nodes. From the measurements, we conclude that the protocol is efficient in the sense that the overhead is negligible when the aggregate is far from the threshold. It is scalable as the protocol overhead is independent of the system size for the network sizes and scenario configurations considered. We demonstrate that the local hysteresis parameter can be used to control the trade-off between protocol overhead and detection delay. We further report results on how node failures impact overhead and detection quality of the protocol.
This paper appears in this thesis as Chapter 8. It is also published as:

> F. Wuhib, M. Dam and R. Stadler, "Decentralized Detection of Global Threshold Crossings Using Aggregation Trees", **Computer Networks**, Vol. 52, Issue 9, pp.1745-1761, June 2008.

My contribution to this work has been the design, implementation and evaluation of the protocol. In addition to fruitful discussions on all aspects of the protocol, Mads Dam is the author of the GAP protocol and he also formalized the protocol invariants and correctness statements.

3.4 A Gossiping Protocol for Detecting Global Threshold Crossings

We investigate the use of gossip protocols for the detection of network-wide threshold crossings. Our design goals are low protocol overhead, small detection delay, low probability of false positives and negatives, scalability, robustness to node failures and controllability of the trade-off between overhead and detection delay. Based on push-synopses, a gossip protocol introduced by Kempe et al., we present a protocol that indicates whether a global aggregate of static local values is above or below a given threshold. For this protocol, we prove correctness and show that it converges to a state with no overhead when the aggregate is sufficiently far from the threshold. Then, we introduce an extension we call TG-GAP, a protocol that (1) executes in a dynamic network environment where local values change and (2) implements hysteresis behavior with upper and lower thresholds. Key elements of its design are the construction of snapshots of the global aggregate for threshold detection and a mechanism for synchronizing local states, both of which are realized through the underlying gossip protocol. Simulation studies suggest that TG-GAP is efficient in that the protocol overhead is minimal when the aggregate is sufficiently far from the threshold, that its overhead and the detection delay are largely independent on the system size, and that the trade-off between overhead and detection quality can be effectively controlled. Lastly, we perform a comparative evaluation of TG-GAP against a tree-based protocol. We conclude that, for detecting global threshold

crossings in the type of scenarios investigated, the tree-based protocol incurs a significantly lower overhead and a smaller detection delay than a gossip protocol such as TG-GAP.
This paper appears in this thesis as Chapter 9. It is also published as:

> F. Wuhib, M. Dam, R. Stadler and A. Clemm, "Robust Monitoring of Network-Wide Aggregates Through Gossiping", **IEEE Transactions on Network and Service Management**, Vol.6, No. 2, pp.95-109, June 2009.

My contribution to the work has been the protocol design, implementation and evaluation. In addition to fruitful discussions on all aspects of this work, Mads Dam from KTH helped with formalizing the protocol and proved the convergence properties.

3.5 Gossip-based Resource Management for Cloud Environments

We address the problem of resource management for a large-scale cloud environment that hosts sites. Our contribution centers around outlining a distributed middleware architecture and presenting one of its key elements, a gossip protocol that meets our design goals: fairness of resource allocation with respect to hosted sites, efficient adaptation to load changes and scalability in terms of both the number of machines and sites. We formalize the resource allocation problem as that of dynamically maximizing the cloud utility under CPU and memory constraints. While we can show that an optimal solution without considering memory constraints is straightforward (but not useful), we provide an efficient heuristic solution for the complete problem instead. We evaluate the protocol through simulation and find its performance to be well-aligned with our design goals.
This paper appears in this thesis as Chapter 10. It is also published as:

> F. Wuhib, R. Stadler, M. Spreitzer, "Gossip-based Resource Management for Cloud Environments", **6th International Conference on Network and Service Management**, Niagara Falls, Canada, October 25–29, 2010.

My contribution to this work has been the design, implementation and evaluation of the protocol. Mike Spreitzer originally suggested addressing the problem of resource management for cloud environments and gave very helpful comments during the development of the protocol and the writing of the paper.

Chapter 4

Open Problems for Future Research

The research in this thesis has revealed a set of issues in distributed monitoring and resource management that merit further investigation.

1. A fundamental question that we did not address in this thesis is *to which extent a given management task in a large-scale network system should be distributed*. We have obtained scalability results for certain metrics, mostly using simulation. However, we believe that a thorough and complete investigation is needed that studies the conditions and the benefits, as well as cost, of centralized vs. decentralized management.

2. Another important question that we have not addressed in this work is *the influence of the overlay topology on the performance of management protocols*. Depending on our goals of investigation, we have used (1) overlay topologies matching the underlying physical topology[91], (2) a grid topology[95], (3) random networks with fixed degrees generated by GoCast[86, 93, 89], and (4) dynamic random networks generated by Cyclon[98]. We believe that a systematic investigation is needed into how specific topological properties of the management overlay affect the performance, scalability, and robustness properties of management protocols.

3. An in-depth investigation is needed in which tree-based and gossip-based management protocols are compared. Our work indicates that tree-based monitoring protocols perform better than gossip-based protocols in many relevant scenarios. Results from a master thesis identify cases (though not likely encountered in practical settings) where gossip-based protocols potentially outperform tree-based protocols with similar functionality[81]. The comparison of tree-based and gossip-based protocols should include analytical modeling and should consider the influence of the evolution of local variables as well as the particular choice of aggregation functions.

4. With respect to distributed resource management, the results reported in this thesis are building blocks towards engineering a resource management solution for large-scale clouds. The following functionalities are needed for a comprehensive solution: (1) a distributed mechanism that efficiently places new sites onto machines, (2) an extension to our middleware design that renders the resource allocation process robust to machine failures and (3) a resource allocation scheme that spans several clusters and datacenters.

5. With respect to the specific protocol we propose for resource allocation, some open issues remain: (1) how our heuristic solution compares to an optimal one and how this difference depends on the memory demand, (2) whether and how other notions of utility can be supported with the approach in this thesis, (3) how to devise a mechanism that allows controlling the trade-off between the cost of reconfiguration and maximizing the cloud utility (4) how to develop a tree-based protocol for resource management in cloud environments and how such a protocol compares with a gossip-based protocol with similar functionality.

Chapter 5

List of Publications with Results from this Thesis Research

1. F. Wuhib M. Dam R. Stadler and A. Clemm, "Decentralized computation of threshold crossing alerts," In Proc. 16th IEEE/IFIP Workshop on Distributed Systems: Operations and Management, pages 220-232, Barcelona Spain, October 24–26, 2005.

2. F. Wuhib, R. Stadler and A. Clemm, "Implementation and Evaluation of a Protocol for Detecting Network-Wide Threshold Crossing Alerts," 4th IEEE/-IFIP Workshop on End-to-End Monitoring Techniques and Services, Vancouver, Canada, April 3 2006.

3. F. Wuhib, R. Stadler and A. Clemm, "Decentralized Service Level Monitoring using Network Threshold Crossing Alerts," IEEE Communications Magazine, Vol. 44, Issue 10, October 2006.

4. F. Wuhib M. Dam R. Stadler and A. Clemm, "Robust Monitoring of Network-wide Aggregates Through Gossiping," In Proc. the 10th IFIP/IEEE International Symposium on Integrated Network Management, Munich, Germany, May 21–25, 2007.

5. F. Wuhib, M. Dam and R. Stadler, "Decentralized Detection of Global Threshold Crossings Using Aggregation Trees", Computer Networks, Vol. 52, Issue 9, pp.1745–1761, June 2008.

6. R. Stadler, M. Dam, A. Gonzalez and F. Wuhib, "Decentralized Real-time Monitoring", 2nd ACM Workshop on Large-scale Distributed Systems and Middleware, IBM T. J. Watson Research Center, Yorktown, NY, USA, Sept 15–17, 2008.

7. F. Wuhib, M. Dam, R. Stadler and A. Clemm, "Robust Monitoring of Network-Wide Aggregates Through Gossiping", IEEE Transactions on Network and Service Management, Vol.6, No. 2, pp.95-109, June 2009.

8. F. Wuhib, M. Dam and R. Stadler, "Gossiping for Threshold Detection", 11th IFIP/IEEE international Conference on Integrated Network Management, New York, NY, USA, June 01–05, 2009.

9. F. Wuhib, M. Dam and R. Stadler, "A Gossiping Protocol for Detecting Global Threshold Crossings", IEEE Transactions on Network and Service Management, Vol.7, No. 1, pp.42-57, March 2010.

10. F. Wuhib, R. Stadler, M. Spreitzer, "Gossip-based Resource Management for Cloud Environments", 6^{th} International Conference on Network and Service Management, Niagara Falls, Canada, October 25–29, 2010.