Challenges and Opportunities of Cloud Computing

Trade-off Decisions in Cloud Computing Architecture

Michael Hauck, Matthias Huber, Markus Klems, Samuel Kounev,
Jörn Müller-Quade, Alexander Pretschner, Ralf Reussner, Stefan Tai

`hauck@fzi.de, matthias.huber@kit.edu, markus.klems@kit.edu,`
`kounev@kit.edu, joern.mueller-quade@kit.edu,`
`alexander.pretschner@kit.edu, reussner@kit.edu, stefan.tai@kit.edu`

Karlsruhe Institute of Technology

Technical Report

Vol. 2010-19

1 Introduction

In recent years, Cloud Computing has become an emerging technology that gains wide influence on IT systems. Cloud Computing is a distributed computing model for enabling service-oriented, on-demand network access to rapidly scalable resources [9]. Such resources include infrastructure as a service (IaaS), development and runtime platforms as a service (PaaS), and software and business applications as a service (SaaS). Clients do not own the resources, yet applications and data are guaranteed to be available and ubiquitously accessible by means of Web services and Web APIs "in the Cloud".

1.1 Value Proposition

The main value proposition of Cloud Computing is to provide the clients a cost-effective, convenient means to consume the amount of IT resources that is actually needed; for the service provider, better resource utilization of existing infrastructure is achieved through a multi-tenant architecture.

From a business perspective, Cloud Computing is about improving organizational efficiency and reducing cost, often coupled with the objective of achieving a faster time-to-market. Centrally hosted services with self-service interfaces can help to reduce lead times between organizational units who use the cloud as a collaborative IT environment. Re-usable components, packaged on virtual machines, provide a way to exchange working IT solutions. Capabilities to allocate and de-allocate shared resources on demand can significantly decrease overall IT spending. Low-cost access to data centres in different geographical regions may further reduce market entry barriers and enable new business models.

From a technology and engineering perspective, Cloud Computing can help to realize or improve scalability, availability, and other non-functional properties of application architectures. In this paper, we focus on the technology perspective, and in particular on challenges and opportunities of Cloud Computing research related to quality-driven software service architectures. These include aspects of availability, runtime performance and power management, as well as privacy and distributed data usage.

1.2 Challenges & Research Questions

Not all desired architectural properties can be achieved at the same time. Trade-off decisions have to be made between several (sometimes contradictory) goals, such as:

- increase availability & reliability
- increase performance (latency, throughput)
- increase security and ensure privacy

Which metrics are useful to describe and analyze these trade-off decisions? Based on specific software architecture styles and solutions: How are these goals correlated? How can trade-offs be accounted for during application design, how can they be adapted during run-time?

Since several cloud providers succeeded to introduce scalable, highly available software components such as messages queues or data stores as a service, building a software application (as a service) to be deployed in the cloud requires new architectural decisions and decision-making processes.

Which cloud services can be adapted as components of a new software/service? How do we differentiate or compare existing services and measure their properties? Which is the right service to be used in the cloud application?

1.3 Virtualization

Virtualization technology provides the technical basis for Cloud Computing. Virtualization has already been in the focus of research in the early 1970s [36], but gained a lot of attention in the last years, as inexpensive servers and client machines became powerful enough in order to be used for virtualization. In general, virtualization deals with the creation of virtual resources, such as operating systems, servers, or storage devices.

Different kinds of virtualization can be distinguished: *System virtualization* adds a hardware abstraction layer on top of the hardware, which is called hypervisor or virtual machine monitor. On top of this layer, virtual machines can be run on the physical machine which run regular operating systems. Virtual Machines do not have direct access to the hardware, as opposed to non-virtualized operating systems. or the hypervisor itself Except for a special domain (Dom0), the hypervisor runs virtual machines in a non-privileged environment (DomU). Using system virtualization, multiple virtual machines, which may run various operating systems, can be run on a single physical machine. In a similar way, *storage virtualization* provides access to a logical storage that abstracts from (possibly heterogeneous) physical storage devices. *Application virtualization* provides a virtualized environment that runs inside an OS process and provides a platform-independent environment for applications (e.g. Java applications running in a Java virtual machine). Other application virtualization approaches exist that allow for executing applications without having to install them (e.g. terminal services).

For Cloud Computing environments, system virtualization is the most important technology that is used to provide IaaS, PaaS and SaaS resources. In this paper, the term virtualization is used in the context of system virtualization.

1.4 Decision Support

When building new software applications or services that might potentially be deployed in the cloud, some decisions are inevitable in different stages of the software engineering process. This starts with initial build-or-buy decisions which lead to subsequent questions of how services are operated and who has control over and holds responsible for service delivery. For example, use of an in-house IT department may be compared to the use of third party service providers. In the context of cloud computing, this implies an important, principle business decision whether to own and maintain a data center or outsource operations to the cloud. The later stages of the decision process include the design, deployment and operations of applications and services.

Optimizing the non-functional properties during design- and run-time, and considering to employ existing scalable, highly available services from the cloud requires decision support methods that help to make the right decisions and build cloud services that fulfil goals and requirements, both non-functional and functional.

Generic approaches to optimization and decision-making already exist [93] [52] and can be applied to build a comprehensive decision support framework. However, customizing these methods to make them applicable to the decisions that have to be made in a cloud computing context is yet an open research issue. A first approach and framework has been presented in [61].

In this paper, we explore the opportunities and critical challenges of Cloud Computing that represent different, potentially conflicting objectives. Understanding these objectives is fundamental to Cloud service engineering. Using frameworks such as [61], optimization techniques or artificial intelligence methods can subsequently be applied to find optimal trade-offs or solutions in decision-making over measured data and for multiple goals. Moreover, when making decisions based on non-measureable data, multi-criteria decision-making methods can help to turn qualitative, non-measurable information into numbers [78]. Using methods to retrieve evaluation results of multiple alternatives regarding many criteria to make alternatives comparable, allows immediate, substantiated decision-making by simple selection.

For the different objectives, i.e. availability and performance of software running in the cloud, as well as questions related to privacy and distributed data usage, we give an overview on the state of the art, highlight important research questions, and outline approaches to tackle the presented challenges.

2 Challenges and Opportunities

Cloud Computing introduces a number of technology and engineering challenges, many of which relate to “traditional” requirements of distributed systems, which now must be revisited in the context of virtualized environments.

2.1 Availability

Generally, availability is the degree to which a system is operable, that is, capable of producing responses to submitted requests. Stronger definitions of availability may include objectives with regard to the time window allowed for any response to arrive, or the time window allowed for the system not to be operable. In modern web environments, *high availability* often is a key requirement, as even the slightest outage can introduce significant financial consequences and impact customer trust. High availability typically is addressed by means of replicating servers and storage. This introduces, however, the need to balance trade-offs with other system properties. In the remainder of this section, we will discuss how to leverage cloud architectures in order to improve availability. We will revisit other aspects of availability in Section 2.4.

2.1.1 State of the Art

The notion of *server farms* and *geoplex* architecture can be used to describe and discuss cloud computing architecture [25]. Server farms are distributed systems based on large-scale server clusters that are located at a single site (a single data center). Server farms that are geographically replicated across different sites and connected by the Internet constitute a geoplex.

Scalability and high availability are key requirements of modern cloud-based applications and services. Incremental scalability can be achieved by partitioning large data sets into data shards which are then distributed across multiple servers. This technique does not improve availability, though. Availability is a major challenge in the face of massive numbers of servers constituting an environment where frequent failures are a fact to be coped with. Replication of servers and storage is the key technique to achieve high availability. Alternatively and additionally, Recovery Oriented Computing (ROC) proposes various techniques to improve availability by focusing on failure recovery instead of failure protection [68].

Replicating data across networked servers provokes a trade-off challenge known as the *strong CAP principle* [30] [35]. CAP states that only two of the three properties, transactional consistency (C), high availability (A), and resiliency to network partitions (P) can be achieved at the same time. In widely distributed systems – typical cloud computing environments – partitions are considered inevitable, leaving the trade-off between consistency and availability. For example, strong consistency can be achieved by pessimistic replication mechanisms at the cost of availability. On the other side, high availability can be achieved by optimistic replication at the cost of consistency. The *weak CAP principle* is a generalization of the strong CAP principle by formulating the trade-off as a continuum in lieu of just binary choices. In particular, relaxing consistency requirements and trading them for higher availability is a hot cloud computing research topic [13] [87] [74].

Due to ease of scalability, key-value data stores have become popular solutions for a wide range of cloud-based applications and services, from Amazon's shopping carts to Zynga's social gaming engine. Amazon developed "Dynamo", a distributed key-value data store with high reliability and availability requirements. Dynamo stores relatively small objects, usually less than 1 MB in size. A key requirement of Dynamo is that the data store is "always writable", i.e. users should never experience a situation in which they perceive a failed write. Amazon uses high-percentile performance-oriented Service Level Agreements (SLAs) to monitor the quality of its system architecture requirements. The architecture is configurable in a way that lets developers (and services) make trade-offs between functionality, non-functional system requirements, and cost.

The Google File System (GFS) is representative for cloud-based file storage. GFS supports many of Google's services, such as web indexing, Google Earth and Google Finance. Different from Amazon's Dynamo and Zynga's Membase key-value stores, GFS is optimized for storing very large files (one file is usually many GBs in size) and specific access patterns, such as large streaming reads and large sequential writes. The architecture of GFS is based on a single Master node and multiple chunkserver nodes. Files are split up into chunks, by default 64 MB in size, and distributed over multiple chunkserver nodes. The Master node coordinates multiple chunkserver nodes and manages file system metadata in memory. Google's software engineers are encouraged to program applications in a way that is compatible with GFS; for example: append records to files instead of overwriting files. Many concurrent append operations can be merged and therefore simplify inconsistency conflict resolution. A copy-on-write snapshot mechanism allows developers to quickly branch copies of large data sets and eventually roll back to a previous state if something goes wrong.

Researchers from Yahoo! struggle with the same challenges as their colleagues at Google and Amazon. Against the background of Internet-scale distributed systems, the PNUTS database system has been developed. PNUTS has been optimized for workloads that read and write only a single record or a small group of records. Similar to Google's GFS-based "Bigtable" data store, PNUTS offers developers a simple database system interface without the complex query functionality of modern relational database systems. Operations are restricted to single-table selection and projection. Referential integrity and other constraints are not enforced by PNUTS; join operations and the like are not provided.

2.1.2 Goals and Approach

Cloud computing is widely perceived as a disruptive technology shift from on-premise infrastructure, platforms andapplications to Internet-centric infrastructure, platform, and software services [53] [57]. Moving on-premise applications and services into the cloud can improve availability, particularly along with the following properties:

- Worldwide access
- Workload elasticity
- Fault-tolerance & disaster-tolerance

Desktop applications and applications deployed in small-scale networks (LANs) are designed to be accessible to a group of local users. Although access can be granted to remote users in different geographic regions, this is not the natural modus operandi of such applications. Moving an application into the cloud can simplify access of worldwide users.

Workload elasticity means adapting system resources to changing workloads in real-time, i.e. growing under increasing workload and shrinking under decreasing workload. Vertical scalability – replacing old hardware with new hardware – is not well-suited for providing workload elasticity. Horizontal scalability, i.e. partitioning and replicating homogeneous system components, on the other side, allows incrementally adapting system resources to changing workloads. It is an open question how traditional applications, based on flat file system storage, XML data stores or relational database systems, can be migrated to incrementally scalable data stores like Dynamo, GFS and PNUTS.

Architectures based on a federation of clouds allow for “cloud bursting” by allocating resources of multiple different cloud environments. Moreover, a federated cloud provides richer capabilities, such as the different application-specific data store solutions discussed before. Users of a federated cloud could for example employ simple key-value stores for caching small web site objects and scalable distributed file systems for storing and retrieving larger multimedia content.

Cloud-based application architecture can provide improved levels of fault-tolerance compared to locally hosted applications – and even disaster-tolerance. Infrastructure as a service (IaaS) allows allocating and de-allocating system resources on demand. Thereby, replication techniques can potentially be implemented more cost-efficiently. Research work on building a ”Recovery Cloud” approaches disaster recovery from the perspective of application migration and configuration management where compute clouds are used as an on-demand “emergency environment” [54]. Cloud computing services and technology can be elevated to provide an inexpensive rapid recovery solution. The approach is based on automated workflows of virtual appliances in a compute cloud, in combination with data backup and recovery mechanisms of a storage cloud.

2.1.3 Research Questions and Challenges

Cloud computing provides techniques and infrastructure for building highly available, Internet-scale applications and services.

- What are the major trade-off decisions when moving into the cloud? The trade-off between “availability” and “consistency” serves as a prominent example. Other trade-off decisions must be identified and evaluated, particularly taking into consideration “scalability”, “reliability”, “performance” (latency and throughput), and “cost-efficiency”.
- Benchmarking cloud services requires a new set of benchmarking tools that consider cloud-specific properties, such as practically infinite scalability, relaxed consistency guarantees, on demand resource allocation and accounting, et cetera.

Cloud computing offers the exciting opportunity for an overhaul of antiquated information systems. However, organizations prefer to re-use existing application code and binaries – which count as assets on the balance sheet. If the information system must be operational at all times, a migration is further complicated. How to build new cloud-based applications that accomplish aforementioned promises of improved scalability and availability? How to migrate existing software into the cloud?

An increasing number of heterogeneous end-user devices demands for connecting the cloud to multiple delivery channels. Applications and services should be delivered to smartphones and netbooks, TV sets, cars, et cetera. On the other side, there are also multiple heterogeneous cloud services, such as data store solutions, compute clouds, and messaging services.

- How can Desktop applications and traditional network-based application architectures, such as Java EE architectures, profit from the integration of cloud services or being moved into the cloud altogether?
- Which new tools are necessary to facilitate service (re-)engineering and migration?
- Heterogeneous cloud services and application delivery channels demand for integration technology. How should a “Cloud Service Bus” be designed?

2.2 Design-time Performance Prediction

Reasoning about the performance of a software system is a key factor that has to be taken into account in software development. If performance flaws of a software are detected early in the software development process, costs and efforts of changing the system to increase the performance are lower compared to later changes. Model-driven performance prediction proposes a solution which allows the software architect to reason about the performance during the design-time. For example, such predictions can be used to assess different software design alternatives with respect to software performance. Based on a design model of the software and the system the software is running on, performance analysis models are derived which can be solved by different analysis tools.

However, performance prediction approaches have to be enhanced to be used for software running in virtualized environments, as it is the case for cloud computing solutions.

2.2.1 State of the Art

Design-time performance prediction of software systems has been in the focus of research for the last years (surveyed in [8] and [55]). However, the recent trends of software running in virtualized environments proposes new challenges to software performance prediction. Reflecting performance-relevant virtualization properties would also allow for choosing between different virtualization solutions, as the choice of virtualization solutions may have an impact on the application's performance. As the performance heavily depends on the hardware resources and the system environment a software is running on, these factors have to be taken into account for performance prediction. Such models are available for analyzing the performance of non-virtualized software systems, but reusing the models for analyzing software running in virtualized and cloud-computing environments may lead to imprecise or wrong prediction results.

In [59], [60], an approach to model the performance impact of server consolidation with virtual machines is presented. The authors use an analytic queueing model to describe the performance (i.e., response time, throughput, utilization) of virtual machines running on a single hypervisor. This approach does not provide means to model the mapping of virtualized resources to physical resources explicitly. The system is abstracted at a very high-level and some basic analytic performance models are used which provide very rough estimates of the system performance. In [42], an extension of the PCM meta-model [10] has been proposed that allows for modeling hierarchical schedulers. While allowing rudimentary modeling of complex execution environments, explicit properties of hypervisors cannot be specified or used for performance predictions.

2.2.2 Goals and Approach

To allow for accurate performance predictions of software running in virtualized environments, performance-relevant properties of the virtualization layer have to be taken into account. However, integrating such properties into an analysis model is a cumbersome task, as it has to be done manually and requires domain knowledge. In addition, different virtualization techniques come with different performance properties, making a reuse of an existing performance prediction model problematic for a different virtualization solution.

To enable performance predictions for virtualized software systems, we propose an approach that allows to detect performance-relevant properties of virtualization environments automatically through goal-oriented measurements. Such measurements are to be conducted on the target platform (semi-)automatically. The results of the measurements are then evaluated with statistical techniques and integrated into the performance prediction model. Developing such an approach with a high degree of automation leads to the encapsulation of domain knowledge and the developing of a performance model with a limited effort. The performance analyst can also repeat the measurements, for example for repeated performance analyses based on updated performance models when virtualization properties have been changed.

We applied the approach to operating system scheduling properties and yielded promising results [41]. To apply the approach to virtualized environments, different performance-relevant properties have to be regarded, and thus different measurements have to be designed.

As measurements are taken on the target platform, additional challenges arise when performing them on virtualized systems or cloud-computing environments. First, usually no full control on the target system is given. Thus, measurements should make as few assumptions on the platform as necessary in order to work for different virtualization solutions. Second, software running in virtualized systems or cloud-computing environments have to share physical resources with other applications. This also means that while performing the measurements, additional load has to be expected which can lead to disturbed measurement results. To derive reasonable performance models from the experiment results, the measurements and measurement techniques have to be designed in a way that they can cope with the possibility of additional system load. Here, existing approaches have to enhanced, as usually a system without disturbing load is assumed.

2.2.3 Research Questions and Challenges

To predict the performance of applications running in virtualized environments during design-time, the following challenges have to be addressed:

- What are the performance-relevant properties of virtualized environments that can be taken into account during design-time?
- How to model software for design-time performance prediction, and how to include performance-relevant properties of virtualized environments into the model?

- How to integrate such performance-relevant properties into performance analyses?

To automatically detect performance-relevant properties of virtualized environments through goal-oriented measurements, additional challenges arise:

- How to design measurements to detect performance-relevant properties in a technology-independent way?
- How to deal with additional load on the system which might lead to disturbed measurement results?

2.3 Run-time Performance and Power Management

Cloud computing and virtualization promise substantial reduction of IT operating costs resulting from higher energy efficiency and lower system management costs. Today, only 12% of x86 server workloads are running in virtual machines, however, by 2013 that number is expected to rise to 61% [15]. However, the adoption of cloud computing and virtualization comes at the cost of increased system complexity and dynamicity. The increased complexity is caused by the introduction of virtual resources and the lack of direct control over the underlying physical hardware. The increased dynamicity is caused by the complex interactions between the applications and workloads sharing the physical infrastructure. The inability to predict such interactions and adapt the system accordingly makes it hard to provide quality-of-service guarantees in terms of performance and availability. Moreover, the consolidation of workloads translates into higher utilization of physical resources which makes the system much more vulnerable to performance and availability problems resulting from unforeseen load fluctuations. To address these challenges novel techniques for run-time performance and power management in Cloud Computing environments are needed. Such techniques should allow to proactively adapt the system to changes in application workloads and resource allocations in order to ensure that performance requirements are continuously met while resources are used efficiently.

2.3.1 State of the Art

In [89], an analytical model is used to solve resource management problems in virtualized environments using deterministic and stochastic optimization algorithms. A meta-model for modeling the virtualization layer in detail is not provided. The authors concentrate on allocating resources dynamically based on a ”Monitor-Analyze-Plan-Execute” pattern which takes into account evolving system workloads. This approach, however, does not provide means to model the mapping of virtualized resources to physical resources explicitly. The system is abstracted at a very high-level and some basic analytic performance models are used which provide very rough estimates of the system performance.

Other works [81, 91] provide models for resource management in virtualized systems, however, they focus on the deployment of virtual machines and do not provide means to predict the responsiveness and capacity requirements of the latter. A run-time model of CPU-bounded web service applications running in a virtual machine is presented in [82]. Black-box models describing the relationship between CPU allocations in virtual machines and response time are derived from experimental data. Some recent approaches to run-time performance management in virtualized environments were proposed in [95, 90, 66, 49], however, they do not provide any support for performance prediction at run-time in order to guarantee service-level agreements.

2.3.2 Goals and Approach

To address the challenges of predictable and efficient resource management in virtualized data centers comprising a Cloud Computing environment, we advocate the development of novel techniques for self-adaptive management of application performance and energy efficiency. The goal of these techniques will be to continuously optimize the performance and energy efficiency of the computing infrastructure by automatically reconfiguring resource allocations in response to changes in application workloads. The developed techniques will be implemented by network controllers installed locally at each virtualized computing center available in the cloud. Local network controllers will establish the request volumes served by the local infrastructure, the set of VMs executed by each server, the request volumes at the various servers, and the capacity devoted for the execution of each VM at each server. Network controllers might also decide to turn servers on or off depending on the system load or to reduce the frequency of operation of servers enabled by the Dynamic Voltage and Frequency Scaling (DFVS) mechanisms implemented in modern servers.

A model-driven approach will be adopted enabling performance prediction both at system design and deployment time as well as during system operation. The goal will be to continuously optimize the system resource and energy efficiency during operation by automatically reconfiguring resource allocations in response to changes in application workloads. To enforce quality-of-service requirements, system performance models developed at design time and augmented with monitoring data collected during operation will be used at run-time to predict the effect of dynamic changes in the system configuration. Some examples of such changes include: changing the amount of CPU time allocated to VMs, pausing or resuming VMs, suspending or restoring VMs, migrating VMs from one physical machine to another, shutting down or launching physical machines, etc.

The specific goals that will be pursued are listed below:

- Extend existing performance meta-models for component-based software architectures (e.g., [10]) to support modeling applications running in cloud computing environments. In particular, new modeling constructs should be introduced for describing virtualized system resources (e.g., servers, CPUs, main memory) and capturing their mapping to physical resources. Performance models should be structured around the system components (hardware and software) involved in processing service requests and should be parameterized to capture dependencies on the service execution context, e.g., usage profile, connections to external services, resource allocations, and middleware configuration parameters.
- Develop meta-models for capturing the energy efficiency of virtualized data center infrastructures taking into account the resource allocations of individual hosted applications as well as the utilization of system resources.
- Develop efficient analysis techniques for solving instances of the meta-models developed as part of (1) and (2). The analysis techniques should provide flexibility in trading-off between analysis overhead and solution accuracy.
- Develop methods and tools for automatic model maintenance during operation through continuous monitoring of the service infrastructure. Both instances of the performance models developed as part of (1) and instances of the energy models developed as part of (2) should be continuously refined and calibrated during operation.
- Develop efficient techniques for self-adaptive system reconfiguration at run-time to reflect dynamic changes in application workloads. The goal is to continuously optimize the energy efficiency of the data center infrastructure while at the same time ensuring that application performance requirements are continuously satisfied.

2.3.3 Research Questions and Challenges

- Can design-time performance prediction techniques be simplified and adapted so that they can be used for performance prediction at run-time?
- At what level of abstraction should services and infrastructure components in a cloud computing environment be modeled to enable predictability at run-time?
- What model solution techniques are suitable for performance prediction at run-time providing a good trade-off between prediction accuracy and overhead?

- What time-scales are reasonable to apply the various online reconfiguration mechanisms?
- What workload forecasting techniques are appropriate to model the evolution of complex workloads composed of multiple usage profiles of independent applications running on a shared physical infrastructure?
- What utility functions are suitable to evaluate the quality of alternative system configurations in terms of their performance and energy efficiency?
- At what level of granularity should application workloads and system components be monitored at run-time in order to detect changes in workloads and operating conditions early enough to be able to proactively adapt the system configuration accordingly while minimizing the monitoring overhead?

2.4 Privacy in Service-Oriented Computing

2.4.1 Privacy Problems Introduced through Service Orientation

Due to its advantages, cloud computing will replace traditional computing in many fields. Clients will not need to maintain a costly computing center and obtain software and computing resources *as a Service*. Buyya et al. even call Cloud Computing the fifth utility (after water, electricity, gas, and telephony) [16].

Despite the benefits Service orientation has to offer, there are inherent privacy problems. By using services, clients lose control over their data. They can not control if their data gets copied or misused on the server and have to trust the Service provider. Current security mechanisms focus on protecting the data from external adversaries, for example an adversary eavesdropping data transfer between the client and the service. The threat of insider attacks persists. Consider for example a system administrator who copies the data stored on the server(s) in order to sell it to a third party. To ensure trust into services we have to protect the data from internal adversaries, too.

Securing a pure storage service is easy. Encrypting the data before uploading it to the server provides a sufficient level of protection, depending on the encryption used. However, in most cases, this prevents the server from performing any meaningful operation on the data. Hence, more complex services require advanced techniques for providing privacy.

Cryptographic methods like *secure multiparty computation* [37, 18] or *private information retrieval* [32] offer strong privacy guarantees and can solve many privacy problems. Especially since a *fully homomorphic encryption* method [33] was discovered in 2009 which allows calculations on encrypted data. These methods offer strong security guarantees. For example fully homomorphic encryption allows to build services where the service provider does not learn anything about the user input. However, due to high communication and computation costs, these methods are not practical and their costs outweigh all benefits of outsourcing software or data. Moreover, on principle it is impossible to provide strong classical cryptographic guarantees for all services that involve more than one user [86].

Nevertheless, to ensure trust into Services, we need methods that provide privacy guarantees. Since achieving classical cryptographic guarantees is infeasible in most cases, these guarantees have to be weaker, yet provide a sufficient level of protection. Moreover, to be accepted broadly the privacy guarantees must be easy to understand.

2.4.2 State of the Art

Cryptography offers methods with strong security guarantees for scenarios involving different parties while one or more party is not fully trusted. There are methods for different security models [85, 26, 56].

It is possible to apply these methods to services in order to enhance privacy. However, due to their complexity they cancel the benefits of outsourcing, in most cases. Since most services rely on databases, there is much literature to the problem of secure database outsourcing. However, all of the proposed approaches have drawbacks. Most of them lack a formalization of the level of privacy provided. In the remainder of this section, we discuss the methods originated from cryptography as well as the rather practical approaches of the database community.

There are cryptographic solutions for two or more parties cooperatively computing a certain function over a set of data without any party learning anything about the input of other parties except what is learned by the output. Using an interactive protocol, these *secure multiparty computations* [37, 18] can thus solve all computation related privacy problems. The problem is that for each party, the computation cost is higher than computing the whole function on the complete input without any other party. This makes the concept of multiparty computation for outsourcing services too expensive and in fact pointless if the client is the only one with private input.

There are encryption schemes that produce ciphertexts with homomorphic properties: consider for example Textbook-RSA [77]. Multiplying two ciphertexts and decrypting the result yields the same result as decrypting the two ciphertexts and multiplying the plaintexts. However, Textbook-RSA is not considered as secure [14] and only supports homomorphic multiplication. For many years the existence of a fully homomorphic encryption scheme that supports addition as well as multiplication was unclear. In 2009, Craig Gentry discovered such a fully homomorphic encryption scheme [33]. This theoretically solves our privacy problem for services involving only one user: the client could simply use the proposed encryption scheme, and the service provider could adapt its service to work on encrypted data using this scheme. However, this is not feasible since the size of the key scales with the size of the circuit of the algorithm which the service calculates.

Cryptography offers a method to retrieve information from an outsourced database without the server learning anything about the information queried [32]. This is a special case of oblivious transfer [84]. However, these methods are also infeasible in most cases: If the database server must not learn anything about a query, the query issued to the database must contain every cell. Otherwise the server learns which cells do not contribute to the result of the query, and thus learns something about the result set, if no special-purpose hardware is involved [51]. The need to iterate over every cell for every query execution makes private information retrieval impractical in most cases.

In contrast to the rather theoretical methods introduced above, the database community came up with many practical approaches for secure database outsourcing. The concept of *Database as a Service* was introduced by Hacigümüs et al. in 2002 [39]. They propose to use encryption to enhance privacy and evaluate several different ways to encrypt data in a database. However, the user has to hand the encryption key to the server for query processing. This is a security risk in an untrusted server scenario.

Since then the privacy aspects of this concept and the problem of a *searchable encryption* got much attention. Most of the proposed schemes either rely either on separating the database [3] or on creating additional index tables to enable execution of SQL over encrypted data [38]. Most of these approaches have in common that they propose a particular architecture for the database service. Although different schemes have been examined extensively, it is either unclear what level of privacy they provide [22, 17, 45], or they have restricting assumptions about the input data [5, 11], or they rely on special hardware [51].

In order to give security guarantees, one has to specify the security model in which the guarentees hold. The security model is an abstraction from reality

2.4.3 Goals and Approach

A major aim of this project is the development of novel methods that provide provable privacy guarantees, yet are efficient enough to be used in a service scenario. In general, these privacy guarantees have to be weaker than classical cryptographic notions, but provide a sufficient level of protection.

Combinations of architectural and cryptographic approaches (c.f. Section 2.4.2) is a promising direction. In [43] we proposed a separations of duties that can be used to enhance the privacy of services. In [46] a further discussion of separations of duties can be found. We want to study what level of privacy can be provided practically using combinations of architectural and cryptographic approaches. This leads to a deeper understanding of how services can be adapted to achieve certain privacy guarantees.

In order to enhance the privacy of services we need to understand methods that intuitively enhance privacy. We already discussed separating services. Consider as another example a database service that should not learn the client's queries. Private information retrieval schemes suggest to query all cells of the database. As already discussed, this does not scale. Adding dummy queries to the real one intuitively enhances privacy. However, it is not clear what level of privacy can be provided using a small amount of dummy queries. For example in order to hide a distribution of queries dummy queries can be inserted. However, some information may still leak, since transforming a distribution into another arbitrary distribution may be to expensive or even impossible (i. e. infinite number of elements in the query language). We want to examine how dummy queries and dummy data can enhance privacy, how they have to look like, and how they can be generated. This can lead to private information retrieval as well as data storage schemes that provide weaker privacy guarantees than classical schemes but are practical.

To provide security guarantees, we need formalizations of the provided level of privacy. On the one hand, these formalizations have to be weaker than classical cryptographic notions, to achieve feasibility. On the other hand, they have to comprise a sufficient level of privacy. Thus, in contrast to classical notions, these new formalizations are focused on practicability and may incorporate assumptions about hardware (i. e. the presence of trusted or secure hardware). We already formulated a novel security notion for outsourced databases, k-IND-ICP [43]. Informally, this notion describes that relations between attribute values must not leak. We showed the feasibility of a outsourced database that fulfills this notion. However, depending on the requirements of the client, this notion may be unsuitable. Therefore, we need further practical security notions that can be adjusted to different protection requirements. This may led to fulfillable formalizations of privacy requirements that can be used in practical Software as a Service environments.

2.4.4 Research Questions

We have identified the following challenges:

- Security guarantees
  - What can practical security guarantees that can be used in a Software as a Service scenario look like?
  - What are achievable protection requirements?
  - How can they be formalized in order to prove the level of privacy a service provides?
- Realization
  - How can security guaranties achieved practically?
  - How can architectural and cryptographic approaches be combined in order to enhance privacy?
  - Are there architectures that favor the realization of privacy guarantees?
  - Can reasonable assumptions about hardware (TPM, USB smart cards) be used to achieve a certain level of privacy that could not be achieved otherwise?
  - How can services be adapted to achieve certain privacy guarantees?
  - What privacy guarantees can be provided for data in services using standard techniques?
  - How can these techniques be combined efficiently?
  - How can the level of privacy not well understood methods provide be formulated?
  - What privacy guarantees can be achieved using dummy data or dummy queries?
  - How can dummy queries and dummy data be generated in order to achieve the best possible practical level of privacy?

2.5 Distributed Data Usage

The different layers of the cloud, including network, infrastructure, services, and applications, do and will host an abundance of data. Generalizing the observations on privacy in Section 2.4, in many cases, providers of this data have a vested interest in controlling or at least observing both the flow of this data through the cloud and the usage of this data once it has been given away. One part of the security architecture of future clouds hence appears likely to provide abstractions and mechanisms for controlling usage and dissemination of data in a distributed, heterogeneous, dynamically changing system.

Today's cloud computing infrastructures usually require customers who transfer data into the cloud to trust the providers of the cloud infrastructure. This trust extends to both confidentiality and integrity of the data. Depending on the value of the data, however, not every customer is willing to grant this trust without any justification.

Companies with strict data protection policies miss transparency and security guarantees of the cloud infrastructures. Moreover, they are unable to determine where their data is stored, or do not understand how the cloud infrastructure is managed. This lack of transparency makes potential costumers of cloud infrastructures refrain from using the cloud, since their data assets have great value and they can not quantify the risk with respect to unauthorized access.

One common approach in service level agreements of (cloud) infrastructure providers to improve data protection is encryption of virtual hard disks and network traffic. However, this approach does not guarantee data protection from malicious or negligent infrastructure providers. Providers have full control of the cloud infrastructure, and are able to configure or modify it in a way that allows them access to the data at any moment.

Thus, in current cloud computing scenarios, service providers have no alternative to trusting infrastructure providers to keep their data secure, and there is no way to verify the integrity of the cloud's hardware/software configuration. Furthermore, the specification of data protection guarantees in existing cloud service level agreements is done on a best effort basis. Service providers are not given any guarantees in case data is released maliciously or accidentally to unauthorized parties.

Existing cloud infrastructures use virtualization techniques using hypervisors (e.g., the Xen Cloud Platform) to transparently allocate resources of physical hosts for a service provider's virtual machines. In theory, security in virtualization solutions is provided by design. This is because virtual machines are completely isolated from one another by the hypervisor. In practice, cloud administrators are still able to load a malicious hypervisor module and access the memory and disk content of the service provider's virtual machines. Furthermore, existing cloud infrastructure customers do not know who has physical or virtual access to their data by using, say, remote management tools. Therefore, malicious infrastructure providers or one of their internal administrators could manipulate the infrastructure to steal or modify the customers' data for their own benefit.

We plan to provide a framework for data-driven usage control in the cloud, i.e., the extension of usage control by data flow detection concepts. We want to enforce usage control properties, or at least detect their violation, not for one precisely specified object (one specific data container), but rather for all representations of the data, i.e., all containers that actually or potentially contain the respective object. If a policy stipulates that a data object be deleted within thirty days, then we want to be able to delete all copies, or representations, of that object as well. These representations exist at different levels of abstraction: as objects in a runtime system such as a Java VM, as window content, as file content, as network packets, etc. Data flow must be tracked within each and across different levels of abstraction, and usage control must accordingly be performed at different levels simultaneously. From this perspective, a second IT system in the cloud is simply one further level of abstraction in itself. Distributed usage control is hence subsumed by this approach.

In the following, we describe how the necessary trust can be enabled. We will concentrate on two problems. One is concerned with software that runs on a cloud infrastructure and that makes it possible to control (or at least detect) the flow of data through a system. The second problem is concerned with securing the integrity of a cloud infrastructure's software and hardware. In order to solve the problem of distributed data usage in the cloud, we propose to secure the second class of systems with the first class of systems.

2.5.1 State of the Art

Enforcement Mechanisms Enforcement mechanisms for requirements such as "delete after thirty days," "do not copy," "notify me when giving away," "at most three copies," etc., have, for a variety of policy languages [27, 1, 6, 65, 94, 44, 23, 88], been implemented at single layers of abstraction: at the operating system level [40, 47, 75], at the X11 level [69], for Java [21, 48], the .NET CIL [24] and machine languages [29, 19, 92]; workflow systems [7]; service-oriented architectures [4]; the level of an enterprise service bus [34]; for dedicated applications such as the Internet Explorer [28] or in the context of digital rights management [2, 62, 73]. From a slightly different perspective, comparable monitors are also investigated in grids where resources are dynamically assigned and freed, and they are considered in the domain of intrusion detection systems.

The reason for this variety of enforcement mechanisms is that the *data* that has to be protected comes in different *representations*: as network packets, as attributes in an object, as window content, etc. In principle, all these representations eventually boil down to some representation in memory, but it turns out to be more convenient and simpler to perform protection at higher levels of abstraction. For instance, disabling the print command is easily done at the word processor level; taking screenshots is easily inhibited at the X11 level; prohibiting dissemination via a network is most conveniently performed at the operating system level; etc.

In sum, there is a plethora of solutions for different layers of abstraction, but there is neither integration nor protection of these solutions.

Securing the Cloud Current research approaches for cloud security mainly focus on either (I) the service providers' VMs or (II) the host system. In the former area, integrity measurements are performed using the cloud infrastructure's support (e.g. hypervisor). The cloud infrastructure itself is not verified in these approaches. In the latter area, research tends to focuses on the integrity measurement of host machines mostly utilizing the trusted platform module [83] to do attestations.

In terms of category I, the work of Schiffman et al. [80] describes a system for integrity measurements of virtual machines running inside a distributed system. In this work, the trustworthiness of the machines only depends on the input given to the virtual machines for processing, not on actions taken on the physical host. Thus, the possibility of a malicious insider modifying the physical host is not catered to. Quynh and Takefuji [76] describe a real-time integrity monitor for Xen virtual machines. They describe which functions current file system integrity tools are lacking and how they improved this using a real-time monitor. They assume that Dom0 is out of reach of an attacker and thus store all information there to enable "violation reporting." Jansen et al. [50] present a solution to the problem of attesting virtual machines by deploying security services in Dom0. This enables the secure creation and execution of virtual machines, including the ability to attest the virtual machine at runtime. An essential part of the solution is the TPM, which enables secure storage of security policies in Dom0.

In terms of category II, the work of Santos et al. [64] is on integrity measurements and attestation of physical hosts that run virtualization software. Attestations are performed when a virtual machine is migrated from/to other physical hosts. They then represent a snapshot of the system at the time of migration. Using this approach, the problem of migrating to an untrusted physical host is solved because all physical hosts migrated from/to are trustworthy at the point of migration. This work is based on Terra [31] which enables the attestation of the physical host itself when demanded by using a trusted virtual machine monitor (TVMM) as hypervisor. Using TPM functionality, the TVMM provides isolation and separation of the virtual machines as well as an additional sealed storage of the attestation values. Thus the hypervisor has the responsibility to provide the hardware level attestations of the system in a trusted way to software running in one of the virtual machines.

Neisse et al. [63] have taken these ideas one step further and presented a system that allows to detect, both at boot time and at runtime, changes to crucial system files, including hypervisors, kernel, kernel modules, configuration files, etc.

Moreover, there has been work on an Integrity Measurement Architecture (IMA) [79] and on the virtualization of TPM devices (vTPMs) [12]. IMA guarantees the integrity of all loaded system components including kernel, kernel modules, system libraries, etc., by checking them at boot time, therefore providing an early establishment of trust. There are however no further measurements after the initial one. The work in vTPMs focuses on providing virtual TPM devices for virtualization customers that share a single physical host and do not want to share the real TPM with a different objective than the work described in this paper.

From a commercial perspective, some companies have published studies that indicate their intent to follow a similar direction as we have done to enable trust in cloud infrastructures. Intel, VMWare and RSA announced [20] that they are collaborating to build a system to measure and monitor cloud infrastructure security. This system uses the Intel Trusted Execution Technology CPU extension, RSA software components to collect the data, a dashboard for security evaluation, and a component developed by RSA to prevent data loss. The system is called *Data Loss Prevention Enterprise Manager*. The cited report does not contain further details about the product being developed.

2.5.2 Goals and Approach

The general problem of distributed data usage control [67, 70] is concerned with the problem of how to manage data once it has been given away. Application domains include privacy, compliance with regulations, data management in distributed business processes, digital rights management, eGovernment, the management of intellectual property and, in general, that of secrets. Typical requirements include "don't disseminate," "notify me when giving away my data," "delete after thirty days," "don't delete within five years," etc. The more distributed a system is, the more complex the challenge becomes. We are convinced that any cloud technology needs support and built-in approaches to provide mechanisms for the enforcement of distributed usage control policies both at the service and the infrastructure levels. Our approach is hence to (1) have enforcement mechanisms that also cater to cross-layer data flows at different levels of abstraction within one single and in-between different IT system; and (2) secure these enforcement mechanisms in the different IT systems using trusted computing technology (which seems to be the least bad of today's available solutions).

Preventive and Detective Enforcement Usage control policies such as "no non-anonymized data may leave this service," "notify owner upon dissemination," "this data may not be sent to an advertisement server," "don't distributed to a call center" can be enforced both by detection and by prevention. Detective enforcement means that policy violations are merely detected at runtime or upon forensic analysis: no precautions are taken to ensure policies are adhered to. This approach is motivated by the underlying trust model; available technology; the unforeseen consequences of interacting with the controlled system when inhibiting rather than observing actions; cost and practicality considerations. In contrast, preventive enforcement aims at ensuring that policies are adhered to. Classical DRM systems are the most popular instance of this enforcement paradigm; whether or not "they work" is out of the scope of this report. Preventive enforcement can be done by inhibition (blocking requests or dropping messages), by modification (replace name and birth date fields by empty strings to the end of anonymization), and by execution (send email notification) [58, 72, 71]. The decision on which kind of preventive enforcement is up to the user. This is because there is usually not "the right way," but rather many right ways. Whether or not preventive or detective enforcement is chosen in a particular case depends on the business model, the trust model, and available technology.

Layers of abstraction When processed by an IT system, data resides, vertically, at different layers of abstraction. At a technical level, these layers include that of the processor and the memory management itself (the system to be controlled is machine code), that of the operating system, that of runtime systems such as .NET or Java virtual machines, that of infrastructure applications such as window managers or file systems, that of data base systems or workflow engines; but also that of networks, desktop applications such as word processors, web services, etc. Depending on the perspective, the same data hence exists, vertically, in different representations: as network packet, memory cell, Java object, window content, file, data base record, text document, etc. The enforcement of usage control policies could of course ultimately be done exclusively at the lowest level, that of machine code. As it turns out, however, it is often simpler to do this at higher levels of abstraction: copy& paste is conveniently disabled at the level of a window manager or within a word processor; deletion of files is conveniently managed at the operating system level; the sending of messages can conveniently be managed within a service bus or a workflow engine. The reason is that at the level of machine code, it is very hard to identify those parts of the code that are concerned with these actions while natural abstractions exist at the other layers. All the above layers exist, horizontally, in multiple forms when it comes to distributed systems in the cloud. However, the horizontal dimension of the problem conceptually just generalizes the vertical dimension: solutions for the vertical dimension very likely directly imply solutions for the horizontal dimension, the "distributed part" of the cloud. We are convinced that data usage control, be it detective or preventive, must be performed simultaneously at many levels of abstraction and across different machines. There is a huge body of work to leverage that we have described in Section 2.5.1.

Data flow Since data exists in different representations, it is not sufficient to control the usage of single representations of data. In addition, one must track the flow of data through a system: on one single machine, if displaying a sensitive data item contained in a file should be avoided, then it is necessary to vertically track the flow of information (1) from the hard disk (2) to the in-memory representation when read by a process (3) to the window manager (4) to the graphics device, thus encompassing many or all of the above layers of abstraction. Analogously, in a distributed system such a smart energy metering system, data flows horizontally from the (1) sensor to the energy provider's (2) backend, (3) billing system, and (4) statistics engine; and then (5) to call centers and third party service providers. Each of these systems is then vertically structured as mentioned above. The notion of information flow can be both possibilistic (some information has flown) and quantitative; the former is theoretically desirable and appealing but turns out to be practically less useful. This is because some information almost always flows. In any case, it is necessary to track the flow of information within one layer (e.g., files are copied), across layers (e.g., a file is read by a process), and in-between different networked systems. We are convinced that usage control in the cloud is impossible if these different layers of abstraction and the different representations of data both within on single IT system (vertical axis) and on several distinct IT systems (horizontal axis, including embedded devices in the internet of things) are not considered. If usage control policies are to be enforced, any architecture of future clouds must provide abstractions for these fundamental concepts in both dimensions.

2.5.3 Research Questions and Challenges

Roughly, we consider the major challenges in this context to include three problem domains that intensely interact one with the other.

Requirements and specification of policies Usage control policies - at different levels of granularity - need to address data, data representations, actions on data or representations, and, related, different systems or services on which the data representations are stored. For instance, profile data in a social network may come as data base record, file, or Java object. It may be stored in file servers, backup servers, web servers, billing systems, etc. While we believe that it is reasonable to assume that the overall number of data representations is finite and not too large, a particular difficulty is given by the fact that the number of possible representations within one system is not fixed because this system will, by interacting with other systems, change. Existing policy languages [27, 1, 6, 65, 94, 44, 23, 88] must be extended so that they are at the same time simple enough so that end-users can understand them, and expressive enough so that possible representations of data in different systems can be addressed.

Enforcement of policies Once policies have been specified, they need to be enforced at the different levels of abstraction, both vertically within one system and horizontally across different systems . This implies the existence of security monitors that (1) track the flow of data through the system and (2) monitor the actions exercised on the different representations of this data, as described in Section 2.5.1. In terms of preventive enforcement, some special care must be taken as far as the robustness of systems is concerned: today's applications usually crash when, for instance, specific system calls return an error code without being executed.

Vertically, we need to connect these different levels, which turns out to be a difficult problem: how do a browser and an operating system know that the image rendered on a screen and stored in, say, a C data structure in the browser is the same as a cache file on the hard disk? Conceptually, the obvious approach is to implement this by means of a bus system with plenty of consequences for performance and security; technically, this turns out to be less than trivial in the general case.

Horizontally, across multiple machines, we need to build a similar bus that maintains the knowledge that several data representations belong to the same piece of data. We would like to stress here that without securing the vertical dimension, securing only the horizontal dimension is a somewhat futile endeavor. The distributed nature of the cloud then requires, in addition to enforcement proper, solutions for policy lifecycle management, deployment, and the like.

Guarantees and certification When enforcement mechanisms are built, it is necessary to (1) constructively make sure that they provide specific guarantees, and to (2) analytically provide evidence to potential users of the system that is equipped with these enforcement mechanisms. Before giving away data, users should be given the possibility to make sure that a specific mechanism is in place and has not been tampered with. Moreover, beyond today's SLAs or CC certifications, users should be given the possibility to check if a service or a subsystem they plan to use, actually does what it is supposed (or advertised) to do, and to not do what it is not supposed to do. Along the constructive dimension, we believe that work in the area of trusted computing is a promising candidates, even though there is a variety of unsolved problems (taking TPM ownership; managing updates; loss of keys; etc.). Along the analytical dimension, we believe that in addition to static analysis techniques, it is necessary to conceive and implement dynamic sampling approaches, including testing, that allow for remote quantitative assessments of systems in the cloud.

3 Conclusions

In this paper, we presented challenges and opportunities of Cloud Computing technology. Such challenges and opportunities deal with the availability or performance of software running in the cloud, as well as privacy and data control. For these research fields, we highlighted the current state of the art, and presented approaches to mitigate the open problems.

We argue that Cloud Computing introduces new trade-off decisions in the context of quality-driven software service architectures. These decisions include trade-offs between service quality attributes, such as availability, distributed data consistency, service runtime performance, and privacy. We envision a structured decision support framework for cloud-based architectures that explicitly addresses these trade-offs.

References

[1] Multimedia framework (MPEG-21) – Part 5: Rights Expression Language, 2004. ISO/IEC standard 21000-5:2004.

[2] Adobe livecycle rights management es. `http://www.adobe.com/products/livecycle/rightsmanagement/indepth.html`, Aug. 2010.

[3] G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-Molina, K. Kenthapadi, R. Motwani, U. Srivastava, D. Thomas, and Y. Xu. Two can keep a secret: A distributed architecture for secure database services. *CIDR 2005*.

[4] B. Agreiter, M. Alam, R. Breu, M. Hafner, A. Pretschner, J.-P. Seifert, and X. Zhang. A technical architecture for enforcing usage control requirements in service-oriented architectures. In *SWS*, pages 18–25, 2007.

[5] G. Amanatidis, A. Boldyreva, and A. O'Neill. Provably-secure schemes for basic query support in outsourced databases. In *DBSec*, pages 14–30, 2007.

[6] P. Ashley, S. Hada, G. Karjoth, C. Powers, and M. Schunter. Enterprise Privacy Authorization Language (EPAL 1.2). IBM Technical Report, 2003. `http://www.zurich.ibm.com/security/enterprise-privacy/epal/Specification/`.

[7] B. Aziz, A. Arenas, F. Martinelli, I. Matteucci, and P. Mori. Controlling usage in business process workflows through fine-grained security policies. In *Proc. Intl. Conf. On Trust, Privacy and Security in Digital Business*, pages 100–117, 2008.

[8] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni. Model-Based Performance Prediction in Software Development: A Survey. *IEEE Transactions on Software Engineering*, 30(5):295–310, May 2004.

[9] C. Baun, editor. *Cloud Computing : Web-basierte dynamische IT-Services*. Informatik im Fokus. Springer, Heidelberg [u.a.], 2010.

[10] S. Becker, H. Koziolek, and R. Reussner. The Palladio Component Model for Model-driven Performance Prediction. *J. Syst. Softw.*, 82(1):3–22, 2009.

[11] M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In *CRYPTO*, pages 535–552, 2007.

[12] S. Berger, R. Cáceres, K. A. Goldman, R. Perez, R. Sailer, and L. van Doorn. vtpm: virtualizing the trusted platform module. In *USENIX-SS'06: Proceedings of the 15th conference on USENIX Security Symposium*, pages 305–320, Berkeley, CA, USA, 2006. USENIX Association.

[13] K. Birman, G. Chockler, and R. van Renesse. Toward a cloud computing research agenda. *SIGACT News*, 40(2):68–80, 2009.

[14] D. Boneh, A. Joux, and P. Nguyen. Why textbook elgamal and rsa encryption are insecure (extended abstract), 2000.

[15] J. Brodkin. Gartner's data on energy consumption, virtualization, cloud. IT world, The IDG Network, 2008. `http://www.itworld.com/green-it/59328/gartners-data-energy-consumption-virtualization-cloud`.

[16] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599–616, June 2009.

[17] A. Ceselli, E. Damiani, S. D. C. D. Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati. Modeling and assessing inference exposure in encrypted databases, 2005.

[18] D. Chaum, C. Crépeau, and I. Damgard. Multiparty unconditionally secure protocols. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 11–19, New York, NY, USA, 1988. ACM.

[19] J. Clause, W. Li, and A. Orso. Dytan: a generic dynamic taint analysis framework. In *Proc. Intl. Symp. on software testing and analysis*, pages 196–206, 2007.

[20] S. Curry, J. Darbyshire, D. W. Fisher, B. Hartman, S. Herrod, V. Kumar, F. Martins, S. Orrin, and D. E. Wolf. Infrastructure security: Getting to the bottom of compliance in the cloud. `http://www.rsa.com/innovation/docs/CCOM_BRF_0310.pdf`, March 2010.

[21] M. Dam, B. Jacobs, A. Lundblad, and F. Piessens. Security monitor inlining for multithreaded java. In *Proc. ECOOP*, pages pp. 546–569, 2009.

[22] E. Damiani, S. D. C. Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati. Balancing confidentiality and efficiency in untrusted relational DBMSs, 2003.

[23] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The Ponder Policy Specification Language. In *Proc. Workshop on Policies for Distributed Systems and Networks*, pages 18–39, 1995.

[24] L. Desmet, W. Joosen, F. Massacci, K. Naliuka, P. Philippaerts, F. Piessens, and D. Vanoverberghe. The S3MS.NET Run Time Monitor: Tool Demonstration. *ENTCS*, 253(5):153–159, 2009.

[25] B. Devlin, J. Gray, B. Laing, and G. Spix. Scalability Terminology: Farms, Clones, Partitions, Packs, RACS and RAPS. *CoRR*, cs.AR/9912010, 1999.

[26] R. Dowsley, J. Müller-Quade, and A. C. A. Nascimento. A cca2 secure public key encryption scheme based on the mceliece assumptions in the standard model. In *CT-RSA*, pages 240–251, 2009.

[27] R. I. (ed.). Open Digital Rights Language v1.1, 2008. `http://odrl.net/1.1/ODRL-11.pdf`.

[28] M. Egele, C. Kruegel, E. Kirda, H. Yin, and D. Song. Dynamic spyware analysis. In *Proceedings of USENIX Annual Technical Conference*, June 2007.

[29] U. Erlingsson and F. Schneider. SASI enforcement of security policies: A retrospective. In *Proc. New Security Paradigms Workshop*, pages 87–95, 1999.

[30] A. Fox and E. A. Brewer. Harvest, yield, and scalable tolerant systems. In *HOTOS '99: Proceedings of the The Seventh Workshop on Hot Topics in Operating Systems*, page 174, Washington, DC, USA, 1999. IEEE Computer Society.

[31] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: a virtual machine-based platform for trusted computing. *SIGOPS Oper. Syst. Rev.*, 37(5):193–206, 2003.

[32] W. Gasarch. A survey on private information retrieval. *Bulletin of the EATCS*, 82:72–107, 2004.

[33] C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC '09: Proceedings of the 41st annual ACM symposium on Theory of computing*, pages 169–178, New York, NY, USA, 2009. ACM.

[34] G. Gheorghe, S. Neuhaus, and B. Crispo. xESB: An Enterprise Service Bus for Access and Usage Control Policy Enforcement. In *Proc. Annual IFIP WG 11.11 International Conference on Trust Management*, 2010.

[35] S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent available partition-tolerant web services. In *In ACM SIGACT News*, page 2002, 2002.

[36] R. Goldberg. Survey of Virtual Machine Research. In *IEEE Computer Magazine*, volume 7, pages 34–45, 1974.

[37] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC '87: Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229, New York, NY, USA, 1987. ACM.

[38] H. Hacigümüs, B. Iyer, C. Li, and S. Mehrotra. Executing SQL over encrypted data in the database-service-provider model. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 216–227. ACM, 2002.

[39] H. Hacigümüs, B. Iyer, and S. Mehrotra. Providing database as a service. In *ICDE '02: Proceedings of the 18th International Conference on Data Engineering*, page 29, Washington, DC, USA, 2002. IEEE Computer Society.

[40] M. Harvan and A. Pretschner. State-based Usage Control Enforcement with Data Flow Tracking using System Call Interposition. In *Proc. 3rd Intl. Conf. on Network and System Security*, pages 373–380, 2009.

[41] M. Hauck, J. Happe, and R. H. Reussner. Automatic Derivation of Performance Prediction Models for Load-balancing Properties Based on Goal-oriented Measurements. In *Proceedings of the 18th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'10)*, 2010. To appear.

[42] M. Hauck, M. Kuperberg, K. Krogmann, and R. Reussner. Modelling Layered Component Execution Environments for Performance Prediction. In *Proc. 12th International Symposium on Component-Based Software Engineering*, 2009.

[43] C. Henrich, M. Huber, C. Kempka, J. Mueller-Quade, and R. Reussner. Technical report: Secure cloud computing through a separation of duties. `https://sdqweb.ipd.kit.edu/huber/reports/sod/technical_report_sod.pdf`, 2010.

[44] M. Hilty, A. Pretschner, D. Basin, C. Schaefer, and T. Walter. A policy language for distributed usage control. In *Proc. ESORICS*, pages 531–546, 2008.

[45] B. Hore, S. Mehrotra, and G. Tsudik. A privacy-preserving index for range queries. In *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*, pages 720–731. VLDB Endowment, 2004.

[46] M. Huber. Towards secure services in an untrusted environment. In *Proceedings of the Fifteenth International Workshop on Component-Oriented Programming (WCOP) 2010*, 2010. to appear.

[47] G. Hunt and D. Brubacher. Detours: Binary Interception of Win32 Functions. In *Proc. 3rd USENIX Windows NT Symposium*, 1999.

[48] I. Ion, B. Dragovic, and B. Crispo. Extending the Java Virtual Machine to Enforce Fine-Grained Security Policies in Mobile Devices. In *Proc. Annual Computer Security Applications Conference*, pages 233–242. IEEE Computer Society, 2007.

[49] R. Iyer, R. Illikkal, O. Tickoo, L. Zhao, P. Apparao, and D. Newell. Vm3: Measuring, modeling and managing vm shared resources. *Computer Networks*, 53(17):2873 – 2887, 2009.

[50] B. Jansen, H. Ramasamy, and M. Schunter. Flexible integrity protection and verification architecture for virtual machine monitors. In *Proc. Second Workshop on Advances in Trusted Computing*, August 2006.

[51] M. Kantarcioglu and C. Clifton. Security issues in querying encrypted data. Technical report, 2004.

[52] R. Keeney and H. Raiffa. Decisions with multiple objectives. *Cambridge Books*, 1993.

[53] M. Klems, J. Nimis, and S. Tai. Do clouds compute? a framework for estimating the value of cloud computing. In *Proceedings 7th Workshop on e-Business*, LNBIP. Springer, Dezember 2008.

[54] M. Klems, L. Shwartz, G. Grabarnik, and S. Tai. Automating the delivery of it service continuity management through cloud service orchestration. In *Proceedings of the 12th IEEE/IFIP Network Operations and Management Symposium (NOMS 2010)*. IEEE, April 2010.

[55] H. Koziolek. Performance Evaluation of Component-based Software Systems: A Survey. *Performance Evaluation*, In Press, Corrected Proof, 2009.

[56] R. Künzler, J. Müller-Quade, and D. Raub. Secure computability of functions in the it setting with dishonest majority and applications to long-term security. In *TCC*, pages 238–255, 2009.

[57] A. Lenk, M. Klems, J. Nimis, S. Tai, and T. Sandholm. What's inside the cloud? an architectural map of the cloud landscape. In *ICSE Workshop on Software Engineering Challenges of Cloud Computing, 2009. CLOUD '09.* IEEE Press, Mai 2009.

[58] J. Ligatti, L. Bauer, and D. Walker. Edit automata: enforcement mechanisms for run-time security policies. *Int. J. Inf. Sec.*, 4((1-2)):2–16, 2005.

[59] D. Menasce. Virtualization: Concepts, Applications, and Performance Modeling. In *Proc. International CMG Conference*, 2005.

[60] D. Menasce and M. Bennani. Autonomic Virtualized Environments. In *Proc. International Conference on Autonomic and Autonomous Systems*, 2006.

[61] M. Menzel, M. Schönherr, J. Nimis, and S. Tai. $(MC^2)^2$: A Generic Decision-Making Framework and its Application to Cloud Computing. *In Procs. International Conference on Cloud Computing and Virtualization (CCV 2010), Singapore*, 2010.

[62] Microsoft. Windows Rights Management Services. `http://www.microsoft.com/windowsserver2008/en/us/ad-rms-overview.aspx`, 2010.

[63] R. Neisse, D. Holling, and A. Pretschner. Enabling trust in the cloud, 2010. Submitted to ASIACSS.

[64] K. P. G. Nuno Santo and R. Rodrigues. Towards trusted cloud computing. In *Proceedings of the Workshop On Hot Topics in Cloud Computing (HotCloud), San Diego, CA*. MPI-SWS, October 2009.

[65] Open Mobile Alliance. DRM Rights Expression Language V2.1, 2008. `http://www.openmobilealliance.org/Technical/release_program/drm_v2_1.aspx`.

[66] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant. Automated control of multiple virtualized resources. In *EuroSys'09*, pages 13–26, 2009.

[67] J. Park and R. Sandhu. The UCON ABC usage control model. *ACM Trans. Inf. Syst. Secur.*, 7(1):128–174, 2004.

[68] D. Patterson, A. Brown, P. Broadwell, G. Candea, M. Chen, J. Cutler, P. Enriquez, A. Fox, E. Kiciman, M. Merzbacher, D. Oppenheimer, N. Sastry, W. Tetzlaff, J. Traupman, and N. Treuhaft. Recovery Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies. Technical report, 2002.

[69] A. Pretschner, M. Buechler, M. Harvan, C. Schaefer, and T. Walter. Usage control enforcement with data flow tracking for x11. In *Proc. 5th Intl. Workshop on Security and Trust Management*, pages 124–137, 2009.

[70] A. Pretschner, M. Hilty, and D. Basin. Distributed usage control. *Commun. ACM*, 49(9):39–44, 2006.

[71] A. Pretschner, M. Hilty, C. Schaefer, F. Schütz, and T. Walter. Usage Control Enforcement: Present and Future. *IEEE Security and Privacy*, 6:44–53, July/August 2008.

[72] A. Pretschner, M. Hilty, C. Schaefer, T. Walter, and D. Basin. Mechanisms for Usage Control. In *Proc. ASIACCS*, pages 240–245, 2008.

[73] A. Pretschner, M. Hilty, F. Schutz, C. Schaefer, and T. Walter. Usage control enforcement: Present and future. *Security & Privacy, IEEE*, 6(4):44–53, 2008.

[74] D. Pritchett. Base: An acid alternative. *Queue*, 6(3):48–55, 2008.

[75] N. Provos. Improving host security with system call policies. In *Proc 12th USENIX Security Symposium*, pages 257–271, 2004.

[76] N. A. Quynh and Y. Takefuji. A real-time integrity monitor for xen virtual machine. In *Proc. Intl. Conf. on Networking and Services*, page 90, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[77] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[78] T. Saaty. *Theory and Applications of the Analytic Network Process - Decision Making with Benefits, Opportunities, Costs, and Risks*. RWS Publications, 2005.

[79] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and implementation of a tcg-based integrity measurement architecture. In *SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium*, pages 16–16, Berkeley, CA, USA, 2004. USENIX Association.

[80] J. Schiffman, T. Moyer, C. Shal, T. Jaeger, and P. D. McDaniel. Justifying integrity using a virtual machine verifier. In *ACSAC*, pages 83–92. IEEE Computer Society, 2009.

[81] B. Sotomayor, K. Keahey, and I. Foster. Overhead Matters: A Model for Virtual Resource Management. In *Proc. 2nd International Workshop on Virtualization Technology in Distributed Computing*, 2006.

[82] M. Tanelli, D. Ardagna, and M. Lovera. LPV Model Identification in Virtualized Service Center Environments. In *Proc. 15th IFAC Symposium on System Identification (SYSID 2009)*, 2009.

[83] Trusted Computing Group. Trusted platform module main specification 1.2, July 2007.

[84] W.-G. Tzeng. Efficient 1-out-of-n oblivious transfer schemes with universally usable parameters. *IEEE Trans. Comput.*, 53(2):232–240, 2004.

[85] D. Unruh and J. Müller-Quade. Universally composable incoercibility. In *CRYPTO*, pages 411–428, 2010.

[86] M. van Dijk and A. Juels. On the impossibility of cryptography alone for privacy-preserving cloud computing. Cryptology ePrint Archive, Report 2010/305, 2010. `http://eprint.iacr.org/`.

[87] W. Vogels. Eventually consistent. *Commun. ACM*, 52(1):40–44, 2009.

[88] W3C. The Platform for Privacy Preferences 1.1 (P3P1.1) Specification, 2005. `http://www.w3.org/TR/2005/WD-P3P11-20050104/`.

[89] X. Wang, Z. Du, Y. Chen, and S. Li. Virtualization-based Autonomic Resource Management for Multi-tier Web Applications in Shared Data Center. *The Journal of Systems and Software*, 81(9):1591–1608, 2008.

[90] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif. Autonomic resource management in virtualized data centers using fuzzy logic-based approaches. *Cluster Computing*, 11(3):213–227, 2008.

[91] S. Yamasaki, N. Maruyama, and S. Matsuoka. Model-based Resource Selection for Efficient Virtual Cluster Deployment. In *Proc. 3rd International Workshop on Virtualization Technology in Distributed Computing*, 2007.

[92] B. Yee, D. Sehr, G. Dardyk, J. Chen, R. Muth, T. Ormandy, S. Okasaka, N. Narula, and N. Fullagar. Native Client: A Sandbox for Portable, Untrusted x86 Native Code. In *Proc IEEE Symposium on Security and Privacy*, pages 79–93, 2009.

[93] K. Yoon and C. Hwang. *Multiple Attribute Decision Making: An Introduction*. Sage Publications, 1995.

[94] X. Zhang, J. Park, F. Parisi-Presicce, and R. Sandhu. A logical specification for usage control. In *SACMAT '04: Proceedings of the ninth ACM symposium on Access control models and technologies*, pages 1–10. ACM, 2004.

[95] X. Zhu, D. Young, B. J. Watson, Z. Wang, J. Rolia, S. Singhal, B. Mckee, C. Hyser, D. Gmach, R. Gardner, T. Christian, and L. Cherkasova. 1000 islands: an integrated approach to resource management for virtualized data centers. *Cluster Computing*, 12(1):45–57, 2009.