Decentralised and Reliable Service Infrastructure to Enable Corporate Cloud Computing

Christoph GERDES[1], Udo BARTLANG[1], Jörg MÜLLER[2]
[1]*Siemens Corporate Technology, Information and Communications,*
Otto-Hahn-Ring 6, Munich, 81739, Germany
Tel: +49 89 636 44193, Fax: + 49 89 636 41423,
Email: {c.gerdes, udo.bartlang.ext}@siemens.com
[2]*Clausthal University of Technology, Julius-Albert-Str. 4,*
Clausthal-Zellerfeld, 38678, Germany
Tel: +49 5323 727 141, Fax: +49 5323 727 149, Email: joerg.mueller@tu-clausthal.de

Abstract: This paper describes a scalable, adaptive architecture for cloud computing. The platform allows a company to benefit from many advantages of low-cost virtualisation, while still providing control mechanisms to ensure security, compliance, reliability and consistency. We assume the utilised hardware is heterogeneous and potentially unreliable. We therefore adopt Internet proven peer-to-peer (P2P) algorithms, i.e., distributed hash tables (DHTs) and gossip algorithms to cope with the dynamics. In order to reduce setup and maintenance costs we present built in self-configuration and optimization algorithms. A flexible execution environment based on spontaneous group formation ensures reliable service execution even in the event of hardware failure. We show the benefits and performance of the proposed platform by a concrete use case and simulation results.

1. Introduction

Recently, the term cloud computing has been receiving attention as major companies like IBM, Microsoft, Amazon, Google, HP, and others declared their respective vision and strategies. However, yet no clear definition of the term “cloud computing” exists. What is common is the notion of a distributed infrastructure capable of providing services or applications in a location and vendor transparent manner over a public network, usually assumed to be the Internet. Often this includes the combination of massive amounts of dedicated computer hardware distributed globally in various data centres to form a “computing cloud”. The data centres themselves are owned and maintained by the service vendors. While some vendors, e.g., Google focus on the massive computing power capabilities [1], others, e.g., Amazon define the benefits in economic terms through on demand service fees and reduction of data centre costs [2].

We identified as one major drawback of rising cloud computing solutions - or any other type of service on demand architecture – the requirement to transfer possibly sensitive data, e.g., prototype specifications or simulation parameters to a third party service provider, i.e., a prospective competitor. Usually, this violates corporate compliance as such great amount of trust cannot be mustered. Hence, the cloud-computing paradigm as sketched above is restricted to only a few applications.

However, for corporate internal simulations and prototype tests, cloud computing seems to be an ideal fit. For example, our research and development requires highly sensitive simulations of large distributed systems. Similarly, test platforms need to scale up to thousands of entities. Most tests and simulations are project-based meaning the infrastructure is allocated only for a short time period and then released to other projects. It is therefore not feasible to buy dedicated hardware for each project. In addition, due to different project requirements it is very desirable to be able to constantly add or remove computer hardware from the cloud. This leads to pursuit of a decentralized infrastructure being able to deal with such dynamics as demanded by the self-x properties of autonomic computing systems [3]. In this paper, we try to close the gap by introducing a peer-to-peer (P2P) software architecture to build a self-configuring, self-healing, and self-optimizing backend to enable corporate cloud computing.

The rest of this paper is structured as follows: In Section 2 we summarize the objectives of our solution. Section 3 outlines the employed methodology and places our solution into the context of related work. Section 4 sketches the overall system architecture and provides a technical description of the major components. Section 5 illustrates the benefits utilising a concrete use case. Section 6 and Section 7 evaluate the platform performance and reliability as well as highlight the business benefits. We conclude by pointing to ongoing and future work in Section 8.

2. Objectives

The primary aim of this paper is to show how the cloud computing paradigm may be adapted to develop a highly scalable and adaptive distributed computing platform for reliable service execution at corporate sites. We propose the novel approach of a P2P backend to ease the building of a corporate site cloud computing infrastructure.

We assume the underlying hardware to be heterogeneous and potentially unreliable. Hence, the P2P solution should be able to adapt to different hardware and operating systems to exploit existing corporate IT infrastructure and reduce total cost of ownership (TCO). It should ensure reliable execution of dynamically deployed services and consistent data management. The platform must offer a generic application interface not only to deploy generic service tasks and necessary artefacts but also to pass policies specifying quality of service (QoS) requirements. All internal processing should be completely transparent to standard users but allow detailed configuration and fine-tuning if required.

3. Methodology

We show that by combining P2P technologies with the cloud-computing paradigm, we can realise a reliable service infrastructure for heterogeneous hardware environments.

The usage of standard techniques offered by sophisticated grid computing platforms, e.g., [4] would be one approach to manage such infrastructure. But completely adoption and deployment of such grid computing platforms may be very complex and combined with high maintenance costs. Our approach differs from standard grid techniques as we focus on the dynamic character of the platform, the ad hoc and self-organizing features and the adaptability to heterogeneous hardware. Furthermore, our approach is more lightweight and focuses on performance through specialization, e.g., we target to conform to interface standards like OSGi and web services wherever possible but use own internal protocols to improve communication efficiency.

As underlying communication platform we built and utilise the P2P Resource Management Framework (RMF, [5]). The RMF maintains a virtual application-independent abstract collaboration space where metadata (so called resources) can be published, searched, subscribed and modified using certain keywords. RMF employs a Distributed Hash Table (DHT) topology as routing mechanism of the P2P overlay network to run on top of arbitrary physical networks. Here, we use Chord [6] in our prototype implementation.

We follow a service oriented system design approach to support dynamic service deployment. Every peer in the network is modelled as a service providing access to the different computational resources of its host. Similarly each peer provides a container to host other services. We motivate a flexible service model optimised for the dynamic domain of P2P systems. In the respective literature the term “service” is often used ambiguously and hence requires clarification. In our case, a service is simply defined as a self-contained computer program that exports its functionality through a well-defined interface. Services can be parameterized with a task and executed thereby producing a result in a predefined format. A service is not bound to a physical location or specific peer. The location of execution is solely determined by the task and the service description and may change dynamically as the state of the cloud changes over time. A deployment is often transient with the peer removing the service once execution is complete.

As already suggested, the service execution platform offers a generic interface to the outside. This includes a generic web service system interface to deploy services. Internally, we introduce the concept of dynamic P2P service groups to enable reliable service execution. This requires mechanisms to consistently set up service groups and to maintain their inherent structure in order to ensure QoS throughout the operation. Subsequently, we present major techniques how the pure RMF has to be extended in order to deal with the raised requirements of a P2P-based reliable service execution platform.

4. Technology Description

This section elaborates our approach on the technical level. First the overall system architecture is introduced and a brief description of major components is given. With the key concepts introduced, the service execution lifecycle is detailed and finally the mechanisms to achieve high availability and reliable execution are described.

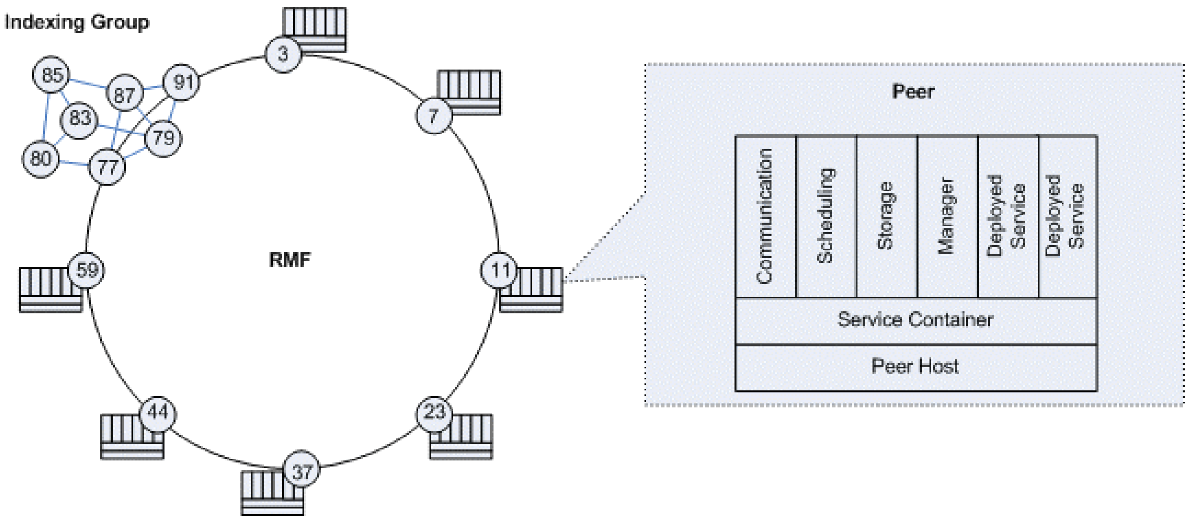

Figure 1: Overall System Architecture of the P2P-based Reliable Service Execution Platform

Figure 1 illustrates the overall system architecture. Each computer node in the cloud is represented by a peer in the RMF. Conceptually, there exist two types of peers: indexing peers and service peers. The reason lies in the nature of DHTs. DHTs provide a simplified interface to efficiently store and retrieve data by keywords; unfortunately they perform miserably for more complex queries such as range queries and semantic queries over large data sets. Therefore, we use indexing peers as a special peer group to concentrate metadata as well as status information of all peers in the RMF. This approach enables the platform to efficiently determine current best peers for service execution, e.g., peers with certain capabilities or idle hardware resources. Indexing peers provide advanced querying interfaces for complex queries required by our internal peer selection process. To enhance communication speed, indexing peers maintain a separate pool of connections to other indexing peers in addition to "normal" DHT connections. The set of indexing peers are referred to as *indexing group*. Metadata may be injected at an arbitrary peer of the indexing group. Afterwards, it is internally disseminated through a gossip-based protocol [7].

Every RMF peer not acting as an indexing peer is considered a service peer, and advertises its resources for service execution. Each peer hosts an OSGi service container with a set of standard services to manage service execution, to provide access to the P2P collaboration space, to store data and to provide scheduling capabilities. Additionally, user services can be deployed into the OSGi service container at runtime.

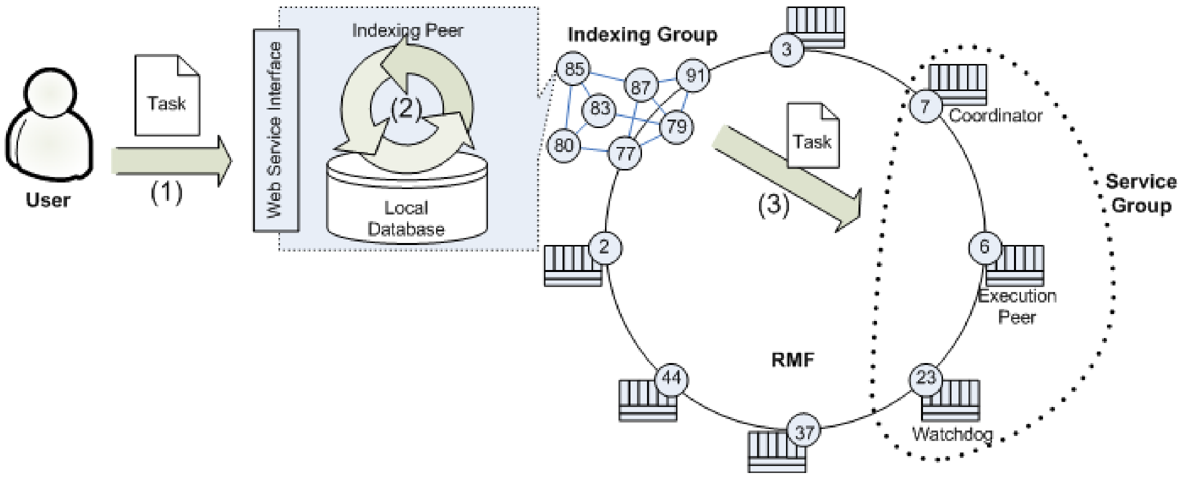

Figure 2: Overview of Service Deployment and Execution

Figure 2 illustrates service deployment and execution process. It is initiated by injecting a description of a particular service task together with a set of policies determining the non-functional requirements of the execution, e.g., demanded processing resources. Tasks are always injected at indexing peers which act as gateways and provide standard web service interfaces (1). An indexing peer then triggers a selection process for possible service group candidates. The selection process applies the task description on the set of peer metadata to filter the peers that are capable of executing the task. Subsequently the set of candidates is further filtered by applying the set of policies. The resulting group is then ranked to find the most suited peers for coordination and execution. The best candidates are determined by their static capabilities (i.e., hardware, network bandwidth etc.), their current status (i.e., load, resources condition) and other criteria provided by the attached policies (2). The next step is to set up a service group configuration. Such configuration consists of a coordinator peer, potentially multiple executing peers plus a set of other peers, so called watchdogs, to act as backups and to monitor the coordinator peer. The number of execution peers, watchdogs, and their monitoring intervals can be regulated via QoS policies (3).

If a peer selected for execution does not provide the service code bundle or other data as requested by the task description, it can be loaded from other DHT peers and dynamically deployed in the local container. Regarding the whole process of publication, look-up, implementation selection and the final loading of platform-specific code we extend former work [8] to deploy OSGi bundles. This is because each function might be available in various implementations with different requirements and properties. This generic and decentralised selection process allows identification of the best-fitting service code bundle for a certain host environment. Here, three categories of properties and requirements have to be fulfilled or at least taken into account during the selection process: the required code bundle service interface, functional and non-functional properties and compatibility requirements, e.g., the required programming language and local execution environment.

Once a service group is formed and execution logic is available, the task description is transferred to the coordinating peer where the logic is parameterized and consequently the execution scheduled using execution peers. During execution phase the coordinator peer acts as group leader and checks with watchdogs (and indexing peers) to maintain the group. At the same time, better-suited execution peers may be discovered once they join the cloud and employed by the coordinator peer.

Execution peers process stateful services by using a check-pointing mechanism to propagate state updates to watchdogs. Thereby the check-pointing interval is controlled by a policy. Should a better-suited peer be found the state may be migrated from a currently executing peer to the new peer.

Process state and execution results are passed back to the user either through the gateway or other location specified by the service. The execution terminates if the service executed the task successfully or a fatal error occurred. In both cases the coordinator disbands the group and resources are reported idle at the indexing group.

Regarding the overall service group operation the central role of the coordinator is crucial. If the coordinator fails during service execution, the group may enter an inconsistent state. Here, we rely on previous work to consistently elect a new coordinator among the watchdogs using a generic consensus protocol instance [9]. This feature is deployed as standard service in the platform. It is able to secure consistency of distributed consensus for benign or even byzantine peer failure models. This provides a superior degree of configurability in terms of fault model and consensus algorithm. Optimal QoS properties may be obtained by application- and environment-specific tailoring of policies. Each of the employed algorithms uses a quorum-based approach to reach distributed consensus between participating peers. In simple terms, progress is guaranteed as long as a quorum of watchdogs is working correctly.

5. Use Case: Large Scale Simulation of Distributed Systems

One particular application of the proposed cloud-computing platform is the large-scale simulation of distributed systems. Large distributed systems are inherently complex developing high dynamics with increasing node numbers. It is therefore essential to simulate and validate such systems with up to millions of nodes.

For instance, the platform constitutes the basis for our distributed simulator. The simulator is constituted by three major components: the simulator code, the simulation code and the simulation parameters. The simulator itself is implemented as an OSGi standard service bundle. The simulation is also packaged as an OSGi bundle. It can be implemented in Java or C code. The third component is an XML formatted text file constituting the task description. The task description includes parameters for the number of execution peers and watchdogs, the number of simulated nodes, simulation runs, result post processing and other simulation related parameters. Additionally some simulations require data files describing network topologies or transmission latencies. These can be downloaded using the storage service or other mechanism specified in the task description. Once simulation bundles are deployed, the coordinator triggers the execution of the distributed simulator. During the execution, the coordinator assigns tasks to the different execution peers and acts as result aggregator. Additionally results and status information are continuously passed back to the user thus enabling real time monitoring of the simulation.

6. Results

We claim a high performance and high availability cloud computing solution. Despite the service to execute, the performance of the complete system depends on the performance of the following key elements: service monitoring, migration as well as checkpointing. In this section we elaborate how each of these components performs given characteristic loads.

High availability is achieved by the watchdog mechanism. The number of watchdogs can be set as required by the task to be executed. Usually an execution group of three peers is sufficient for most applications. Highly dynamic environments such as the public Internet exhibit high churn rates [10], i.e., very short average online times of individual nodes of only a few minutes. This is not realistic in our scenario as we can safely assume that peers in the target environment have online times of at least a couple of hours. However, using 30 minutes as suggested in related work as lower bound we can easily show the high availability of services in the platform. Given an average online time of 30min and average task execution time of 1min the probability for an execution failure is

$$P = \frac{1\min}{30\min} = 0.03$$

in case no watchdogs are used. With watchdogs the failure probability can be further decreased such as

$$P = \frac{1\min}{30\min} \cdot \left(\frac{1\min}{30\min} \right)^k$$

where k is the number of watchdogs. We can achieve an availability of 99.999% with only two watchdogs. The formula assumes the failure of watchdogs is not correlated. This is supported by peer groups being assembled independent of the physical location of the peers.

We did extensive simulations to evaluate the performance of the system regarding our use case. The test cases affected nine Intel Pentium IV type commodity PCs clocked at 2,8 GHz and equipped with 512MB of RAM. All machines were networked through a switched gigabyte ethernet. The operating system was SuSe Linux version 10.1.

A key aspect of distributed simulations is state migration and check-pointing as they are critical for consistent and fast execution take over. Our simulations show migration times of clearly less than $\Delta t_1 = 1sec$ for states of size 10k when migration is triggered due to discovery of a better suited peer. The migration times triggered for the case of unanticipated peer failure are slightly longer i.e. less than $\Delta t_2 = 1sec + 0.5 * monitoring\_interval$.

Figure 3 depicts the scalability of the simulator use case described in the previous section for different service group sizes. Utilizing eight execution peers we are able to simulate more than a total of 500 000 nodes. Thereby the simulation included a complete P2P protocol with peer discovery, resource lookup and stabilisation. The graph shows the number of simulated peers scaling sublinearly with the number of compute nodes, which is due to the fact that the algorithm utilises optimisations requiring slightly memory and compute nodes for larger peer groups. However, being able to simulate distributed systems of this size on comparatively low-end commodity hardware demonstrates what potential lies in the introduced techniques.

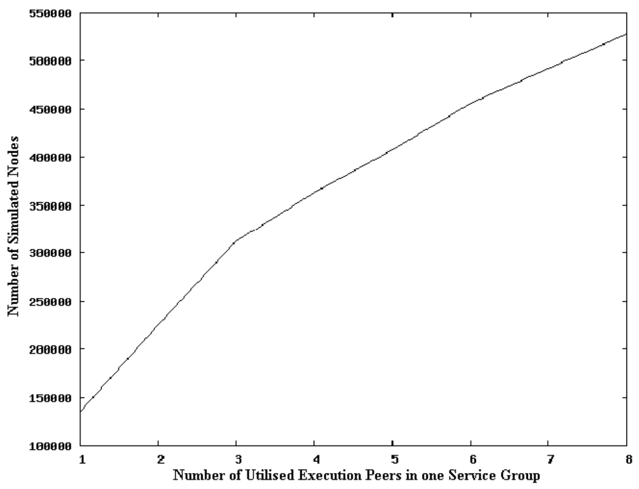

Figure 3: Number of Simulated Nodes Related to Available Hardware Resources

7. Business Benefits

The proposed platform is designed to enable flexible, secure and cost effective cloud computing for corporate installations. Therefore, we use a P2P backend, which is inherently self-organizing thus reducing setup and maintenance costs. Peers can be added to the cloud by simply installing a software package; no further manual configuration is required. This is a major benefit compared to maintenance efforts of static client-server architectures. The platform can run as a software layer on heterogeneous hardware from commodity PCs to high-end servers. The cloud can dynamically extend to new nodes as they become available. It therefore scales seamlessly from small to very large installations eliminating the need to buy expensive monolithic systems. Hence, the P2P approach facilitates cost issues, as the platform can be dynamically scaled up on project demand.

Originally cloud based service execution is entirely location transparent. However, by letting the user specify service execution policies, the user is in full control of where his data is stored and who as access to it. In contrast of other cloud computing approaches, this allows handling of sensitive data in a corporate context.

Energy efficiency is an important topic for data centre operators. The policy based execution system outlined in this paper can be extended such that services are executed on nodes that have advanced power management systems or are physically separated thus producing less heat thus saving energy for air conditioning. This not only reduces operational costs but also constitutes a step towards green computing.

8. Conclusions and Future Work

We described a scalable, adaptive architecture for cloud computing. It lets a company benefit from many advantages of the low-cost virtualization yet still provides control mechanisms to ensure security, compliance, reliability and consistency. We assume the utilised hardware is heterogeneous and potentially unreliable. We therefore adopt Internet proven P2P algorithms, i.e., DHTs and gossip algorithms to cope with the dynamics. In order to reduce setup and maintenance costs we present built in self configuration and optimization algorithms.

This paper motivates and shows how the adoption of the P2P paradigm may facilitate the implementation of a cloud computing backend infrastructure, which we view as an essential building block of a future corporate internet of services. We highlight concepts of adaptive and dynamic service deployment to efficiently realize such infrastructure on top of a basic P2P system.

As we focus on an infrastructure at corporate site we neglect security issues in this paper. For instance, such issues are addressed by Grid Computing which aims at a decentralized infrastructure build for the “open” Internet. Future work is to address advanced policy based security mechanisms to extend the platform for inter corporate collaboration. One meaningful step into this direction may be the adoption and integration of our already developed onion routing solution to enhance anonymity of established collaborations [11].

References

[1] Google App Engine, http://code.google.com/appengine/

[2] Amazon Simple Storage Service (Amazon S3), http://aws.amazon.com/s3/

[3] Jeffrey O. Kephart and David M. Chess. The Vision of Autonomic Computing. IEEE Computer Society Press, 2003, 36, pages 41-50

[4] The Globus Toolkit, http://www.globus.org/toolkit/

[5] Steffen Rusitschka and Alan Southall. The Resource Management Framework: A System for Managing Metadata in Decentralized Networks Using Peer-to-Peer Technology. In Agents and Peer-to-Peer Computing, Springer Berlin/Heidelberg, 2003, Volume 2530/2003, pages 144-149

[6] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In Proceedings of the 2001 ACM SIGCOMM Conference, 2001, pages 149-160

[7] Stephen Boyd, Arpita Ghosh, Balaji Prabhakar, and Devavrat Shah. Gossip algorithms: design, analysis and applications. In INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. (13-17 March 2005)

[8] Rüdiger Kapitza, Holger Schmidt, Udo Bartlang, and Franz J. Hauck. A Generic Infrastructure for Decentralised Dynamic Loading of Platform-Specic Code. In Jadwiga Indulska and Kerry Raymond, editors, Distributed Applications and Interoperable Systems, volume 4531/2007 of Lecture Notes in Computer Science, pages 323-336. Springer Berlin / Heidelberg, 2007. Proceedings of the 7th IFIPWG 6.1 International Conference (DAIS 2007, Paphos, Cyprus, June 6-8, 2007).

[9] Hans P. Reiser, Udo Bartlang, and Franz J. Hauck. A Reconfigurable System Architecture for Consensus-Based Group Communication. In Si-Qing Zheng, editor, Parallel and Distributed Computing and Systems (PDCS 2005), pages 680-686. ACTA Press, 2005. Proceedings of the 17th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS, Phoenix, AZ, Nov.14-16, 2005).

[10] Daniel Stuzbach and Reza Rejaie: Towards a Better Understanding of Churn in Peer-to-Peer Networks, Tech Report, Department of computer science, University of Oregon, November 2004.

[11] Fabian Stäber, Udo Bartlang, and Jörg P. Müller. Using Onion Routing to Secure Peer-to-Peer Supported Business Collaboration. In P. Cunningham and M. Cunningham (eds.), Exploiting the Knowledge Economy: Issues, Applications and Case Studies, Volume 3, pp. 381-188, IOS Press, 2006.