Dynamically Scalable Architectures for E-Commerce

A Strategy for Partial Integration of Cloud Resources in an E-Commerce System

Georg Lackermair[1,2], Susanne Strahringer[1], Peter Mandl[2]

[1]Chair for Business Informatics, Technical University Dresden
[2]Chair for Business Informatics, University of Applied Sciences Munich

1 Motivation

E-commerce became an important economic factor during the last years. Approximately 38.5 million customers in Germany have already bought goods or services online. Furthermore the total revenue in Germany is expected to rise from 46 billion Euro in 2006 to 145 billion in 2010. Increasing mobile web usage and the trading firms' expanding international activities boost this development even further (Stahl et al. 2009).

This growing acceptance of e-commerce will result in an increasing number of both transactions and customers. Considering the extensive and increasing usage of search agents, web-crawlers and interactive elements in addition, workloads in e-commerce will rise significantly. Moreover, the advancing integration with other systems, like other shops requesting catalogue data, causes additional requests to handle.

At the same time, intense competition among retailers tends to result in more demanding requirements with respect to functionality, usability, availability and performance. Rising workloads and more demanding requirements are nowadays handled with statically scaled systems, where the capacity is determined by the expected maximum workload. This results in low resource utilization in data centers conflicting with rising energy costs and CO_2 emissions.

Fluctuation in workloads impairs the situation even further. After Microsoft released Windows 7, various news channels and blogs reported that many online shops had serious performance problems and some servers broke down completely (Computerbase, 2009; Chronicle, 2009). This highlights the problem that workload forecasts are difficult and that surges of visitors can exceed the expected maximum workload significantly. An examination of a B2B shop system showed that even in this domain the workload fluctuation is an issue. The Windows 7 example points out that forecasting workloads and ad hoc extension of computing power pose problems still to be solved.

A solution to these problems could be provided by a strategy for partial integration of cloud computing resources in e-commerce systems. There are many different definitions of cloud computing (Armbrust et al. 2009; Hayes, 2008). In our work we focus on the usage of virtualized computing capabilities on the internet, which is also called utility computing or Infrastructure as a Service (IaaS). Turban et al. (2008, p. 19-13) define utility computing as "computing power and storage capacity that can be used and reallocated for any application—and billed on a pay-per-use basis." A strategy for a partial usage of such virtualized resources on the internet can be realized by implementing a flexible architecture that allows dynamic scaling on the application level. Such an architecture would allow to add resources from the cloud to an online-shop dynamically and on-demand.

After referring to the underlying research approach the paper discusses a partial cloud strategy and provides an example calculation showing its benefits. Furthermore, a general architecture is sketched for implementing this strategy into a system. Finally a flexible architecture is proposed for a shop to support integration of cloud resources on the basis of the general architecture.

2 Research Approach and Scope of the Paper

The underlying overall research project is based on a design science approach as suggested by Hevner et al. (2004) and Hevner (2007). The paper at hand focuses on motivating the relevance of the addressed research problem and suggests a first sketch of an architecture that might help solving this problem. As design science "is inherently iterative" (Hevner et al. 2004, S. 87) and is organized along several design and test cycles our first step is to present the results of a first design cycle. In a next step a prototypical implementation will allow effective evaluation. In order to derive test data for a first evaluation and to further substantiate the relevance of our research we are currently analyzing workload data of a series of online-shops. Some preliminary results are already included within this paper to back up our argumentation but are not the main focus here.

3 A basic E-Commerce System

Laudon and Traver (2009, p. 10) define e-commerce as "digitally enabled commercial transactions between and among organizations and individuals." According to Illik (2002, p. 131ff) an online-shop is the central system in e-commerce.

Figure 1 depicts a basic architecture of today's e-commerce systems, which is derived from our own analysis of typical systems and which is also consistent with Turban et al. (2008, p. 19-3ff). The system usually consists of various interconnected subsystems. An online-shop is the core component that implements the processes for selling goods and services and presents the system to users and customers. It is mostly realized as a multi-layered web-application which is accessed by a web-browser.

An online-shop obtains its master data from a content management system (CMS) and additional information like available stock of an article are fetched from the enterprise resource planning (ERP) or inventory management system. Tracking data is sent to a tracking system; orders are forwarded both to the ERP system and, if needed, to a payment provider. Besides that, a shop system can have dependencies to further systems, i.e. an external logistics system.

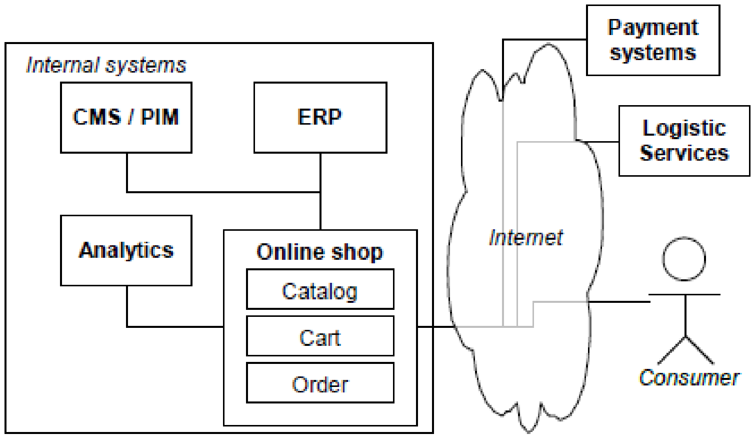

Figure 1: The Subsystems of a Basic E-Commerce System

The basic components of an online-shop are described below:

- The *Cart* component holds a list of articles which the user intends to buy. This information is usually stored permanently.
- The *Catalog* component presents the offered articles to the user. Articles contain descriptive texts, pictures and various attributes. It is common to assign the products to categories.
- The *Order* component implements the process of starting the business transaction. The component forwards orders to the ERP system which initiates the processing of the transaction.

The user interface (UI) is usually presented by means of a web-browser which runs on a client machine. The browser communicates with the server-side by sending requests to a dedicated web server, which forwards the requests for further processing and delivers the results back to the browser. The processing logic can be replicated and usually runs in application servers. Persistent data is stored in a database cluster.

4 Suggested Strategy

For small- and medium-sized businesses (SMBs) it is quite a difficult task to implement dynamic scalability into a web-based system. Running the whole application on own hardware would scale quite well, when it comes to static scaling of a system. Yet its downside is revealed, when an application is to be scaled down after a peak or in times with just a little workload, as owned hardware generates costs, even if shut down.

Therefore, outsourcing of the infrastructure by using cloud resources seems to be appealing, because in the cloud the computing capabilities can be adapted easily and dynamically. But the usage of external service providers has some major drawbacks: A survey among CIOs in Germany pointed out that security concerns and probably evolving dependencies seem to be the most important reasons against the usage of computing capabilities via the internet (TecChannel, 2009). The mentioned dependencies refer to the costs of changing technology and providers, which is described as "lock in" by Shaprio and Varian (1998, p. 106). Hayes (2008) emphasizes transparency issues and Quality of Service (QoS) as major concerns when it comes to Cloud Computing. Armbrust et al. (2009) provide a comprehensive overview on problems involved in using this approach – in contrast to the other references, this work points to the problem that providers operate on a heterogeneous legal basis.

An appropriate strategy must take these concerns into account: Critical components like the frontend web server and the database server may remain in-house, while parts of the application are deployed remotely in virtual machines. The needed infrastructure will be allocated from Infrastructure as a Service-providers (IaaS) who offer virtualized machines for rent. To address the major concerns connected to Cloud Computing, a suitable IaaS-provider must meet several requirements.

From a business perspective it is important, that the pricing model is based on a low level of fixed costs. As the instances in the cloud will be online only for small periods of time, higher variable costs are acceptable, while low or no fixed cost are desirable. Preferably an instance is paid for the hours online, without any base fee. From a technological perspective, there should not be any restrictions about the system that is running inside a virtual machine hosted by an IaaS-provider. There could be restrictions on the operating system or installed software which could cause migration problems and finally would contribute to a lock in. In addition to this, the interface for activating virtual machines should be publicly accessible and standardized. This again minimizes switching costs. If those requirements are met, a multi-sourcing strategy could be implemented, which adds flexibility and decreases dependencies on a single provider (Wannewetsch, 2007, p. 150).

In the following we discuss how the major concerns related to Cloud Computing can be addressed in detail.

5 Addressing Cloud Computing Concerns

In the first run, we focus on the basic technical concerns about Cloud Computing, namely evolving dependencies on a provider, QoS-problems and transparency.

5.1 Evolving dependencies

Some approaches like Google App Engine[1] require that the hosted application is programmed against a proprietary API. Furthermore, as data is stored on proprietary systems SMBs could fear to run into migration problems, once they intend to change the provider and find the data locked (Armbrust et al., 2009). Therefore, the components to be distributed in the cloud should be programmed against standardized APIs and should not store data locally (see 5.3).

For the distribution of an application across network borders, the affected components must be able to integrate dynamically into an application – the configuration may change quite frequently, as the distributed components might be deployed on different nodes, hosted by varying providers. Besides a dynamic link in, standardized mechanisms for invocation of those components, e.g. Web Services (Alonso et al., 2004, p. 152ff) should be used.

5.2 Quality of Service

In the described scenario it is assumed that components in the cloud are added and removed frequently. Furthermore network connection between such a component and other system components can fail. Traditional clustering therefore provides state replication to handle server breakdowns. Across network boundaries, however, this does not seem to be reasonable, as the connections between the cluster nodes are slower, less available and hardly predictable with respect to reliability and latency. As a consequence of this distributed components should be stateless, as such components are easier to recover or migrate in failover situations (Brown, 2008, p. 525ff).

5.3 Transparency

Small- and medium-sized companies might want to keep control over persistent data and keep the backend system in-house. According to Kaufman et al. (2002) privacy of information is a very important aspect of data security. Therefore, data should either be encrypted or not stored at all on machines in the cloud.

The components that are running on machines in the cloud should meet the same security level as components that interact within a closed network. Caching supports this requirement as this makes persistent storage outside trusted data

[1] http://code.google.com/appengine/

centers redundant. Furthermore, all communication with the distributed components should be encrypted, e.g. with VPN tunnelling. In addition to this, cloud components' access rights on backend systems should be limited to read-only access.

6 An Architecture for Dynamic Scaling

An architecture for a partial integration of external computing capabilities is depicted in Figure 2.

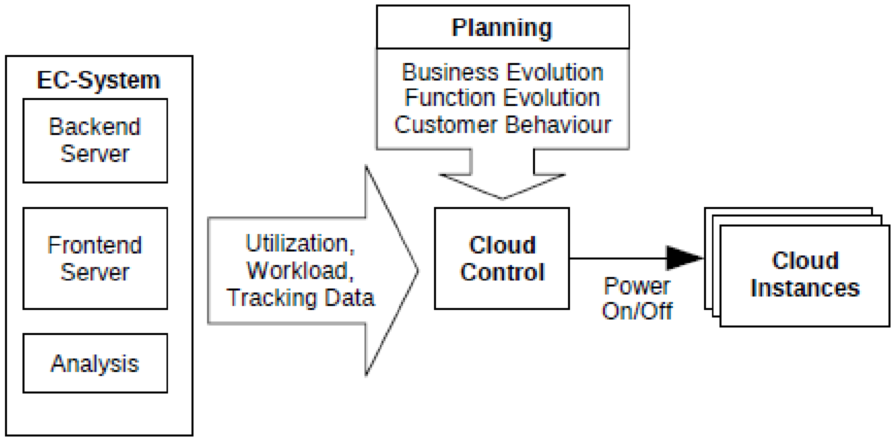

Figure 2: An Architecture for Dynamic Scaling

The application needs to contain a component which controls the cloud instances. Basically, this means to switch on and shut down the instances. An instance has to register as an available node, when the start-up is finished. The cloud control also decides when to add a new node or when to remove a node from the application. Therefore, this component must process information from various sources almost in real-time. Methods and models described by Menascè et al. (2000; 2004) provide a rich set of approaches for capacity planning. However, the concepts still need to be adapted to the specific situation of an automatic planning process.

From the architectural sketch, the most important functional requirements can be derived:

- *Link in:* When a new node is added to the application, the superordinate node, e.g. the balancer needs to get notified about this event to direct workload to the new node.
- *Migration:* When the decision is made to shut down a node, a mechanism must make sure that remaining sessions get migrated to nodes that remain active.

- *Monitoring:* The cloud control needs to monitor subordinate nodes to get information about the resource utilization. Furthermore, a trend analysis of workload as well as ad-hoc-analysis of tracking data help to indicate workload evolution for the short term.
- *Planning:* For forecasting of peak situations planning information needs to be processed. This information is used for long term workload evolution.

The implementation of the link-in and migration mechanisms seem to be technically feasible, as link-in could be based on common presence protocols like XMPP[2]. Existing migration mechanisms for fail-over and load-balancing could possibly be adapted. For monitoring it has to be decided, which information about resource utilization, workload evolution and tracking is available and how it can be processed automatically. For determining the resource utilization all nodes could send information about CPU utilization or waiting processes to the cluster control. Furthermore, the web server could send alerts in case of a significant rise or fall of incoming requests. Ad-hoc analysis of tracking data could help to detect patterns in which peak situations arise.

Processing of planning data can be used to predict times in which peaks or minimal workload are likely. For example, after the introduction of a new product it might be probable that the number of visitors rises significantly. It has to be examined, which information can be provided by different plans and how the data must be provided to be able to process the information computationally.

7 Example: Amazon Elastic Compute Cloud

In order to estimate the economic benefits of a dynamically scaling architecture, total costs of ownership (TCO) for the first year are calculated. The calculation is based on the pricing model for Amazon Elastic Compute Cloud[3], which is shown in Table 1. For sake of simplification the approach of using market prices for renting virtual machines seems to be fairly realistic.

Table 1: Pricing Model for Amazon Elastic Compute Cloud

	On-Demand	Reserved	BEP
Small	0.10$ per hour	325$ + 0.03 $ per hour	4643 hours
Large	0.40 $ per hour	1300$ + 0.12 $ per hour	4643 hour

Amazon offers two different kinds of virtual machines: Small and large instances, whereas a large instance has approximately four times computing capabilities of a small instance. Therefore the example compares the flexible usage of four small

[2] http://xmpp.org

[3] http://aws.amazon.com/ec2/

instances to a single large instance. An instance can be rented on-demand, which means that the customer pays per hour an instance is running. A reserved instance can be rented for a year for a base fee plus a relatively small hourly rate. The Break-Even (BEP) is at 4643 hours. Another interesting aspect is that a large instance causes exactly four times the costs of a small instance.

For static scaling a large instance will be constantly running and therefore the price function can be defined as

$$\mathrm{p_s} = 0.12\$ \times \mathrm{h} + 1300\$$$

where *h* means the number of hours per year. In contrast to this a dynamically scaled system can be composed of four small instances. One instance needs to provide constant availability, because it contains the web server and the cloud control component. Due to the pricing model, a reserved instance is most reasonable for this purpose. The additional instances are added on-demand. The price function can be derived as

$$\mathrm{p_d} = (0.03\$ \times \mathrm{h} + 325\,\$) + \sum_{i}^{n} 0.18 n_i h_i$$

where *n* is the number of utilization classes, which classifies the overall resource utilization of all available computing resources. The number of hours that the system's utilization is classified in utilization class *i* is defined as $\mathrm{h_i}$.

For dynamic scaling some utilization classes have to be defined:

$$\begin{aligned} &h_{25} : 0 < U < 0.25C \\ &h_{50} : 0.25C \le U < 0.5C \\ &h_{75} : 0.5C \le U < 0.75C \\ &h_{100} : 0.75C \le U < C \end{aligned}$$

where *U* means the resource utilization and *C* the overall computing capability, i.e. CPU capacity. The utilization class h_{25} means the number of hours per year in which the overall computing capabilities are utilized up to 25 %, whereas h_{50} means a resource utilization that is equal or greater than 25 % but less than 50 %, and so on. As we assume equal computing power in every node, each load class represents a certain number of small instances needed to handle the workload. In this example h_{25} demands one – the constantly available reserved – instance, h_{50} two, h_{75} three and h_{100} four. This means that for h_{50} one additional on-demand instance is needed, for h_{50} two, for h_{75} three and for h_{100} four.

Table 2: Comparison of Static and Dynamic Scaling

Utilization profile				TCO calculation		
h_{25}	h_{50}	h_{75}	h_{100}	p_s	p_d	delta
20 %	50 %	20 %	10 %	2351.20 $	1639.00 $	-30.29 %
30 %	40 %	20 %	10 %	2351.20 $	1551.40 $	-34.02 %
40 %	30 %	20 %	10 %	2351.20 $	1463.80 $	-37.74 %
50 %	35 %	10 %	5 %	2351.20 $	1201.20 $	-48.92 %

The results of various sample calculations are shown in Table 2. The workload profiles in the table are assumptions based on our own observation of an online-shop. The relative low average workload is consistent to Armbrust et al. (2009, p. 10) who suggest that "for many services the peak workload exceeds the average by factors of 2 to 10". Even if the workload profiles are estimated, the calculation reveals the potential efficiency of the proposed strategy. Moreover, first results of an analysis of a B2B e-commerce system indicate that assuming a relative low mean CPU utilization is realistic. We analysed CPU-utilization of an application server that logged every ten minutes over a period of 23 days and the classification of 3289 data rows has the following distribution:

$$h_{25} = 73.0\%$$
$$h_{50} = 26.6\%$$
$$h_{75} = 0.3\%$$
$$h_{100} = 0.0\%$$

As the table shows (see column *delta*), for a relative low mean CPU utilization, dynamic application scaling would result in significant cost savings.

8 Application Architecture

This part describes necessary changes in a shop's architecture to support the integration of cloud resources. This sketch is mainly based on the principles of service-oriented architectures (SOA). For our work we reduce the high-level approach of Richter et al. (2005) to the technical level, which implies loosely coupled components, e.g. Web Services as described in Turban et al. (2008, p. 19-17ff).

In our scenario, it is assumed that service lookups do not occur on high frequency – a server instance looks up the needed services on start-up and later just on failure of the known provider. Thus, peer-to-peer-based SOA is proposed, as longer latency times for service lookups are acceptable. Nevertheless, any centralistic SOA approach would also be appropriate. Moreover, a dedicated service registry becomes redundant, which reduces the systems' dependency on the availability of such a component.

8.1 Shop-Instances

As mentioned in section 5, shifting to an extensively cached application is necessary for applying such an architecture. Several solutions for distributed caching already exist. Regarding the assumed fluctuation of server instances, for the described scenario, a P2P network could be used as a distributed cache mechanism, as P2P protocols are self-organizing and therefore can handle fluctuation in a network quite well. Whenever an application instance starts, it retrieves a huge amount of catalogue data from a database. A server could share its cached contents with other server instances, without involving the database server during the start-up. The new node will retrieve content data from an already running node. This prevents a dynamically scaled system from causing additional workload on the database server, which is in general considered to be the bottleneck in e-commerce applications (Zhang et al., 2004)

Furthermore, self-organized caching will reduce the workload of the backend systems, as querying for the initial instantiation of the caches descent as well as cache updates do not necessarily need to be propagated to every server instance.

To meet the security requirements defined in section 5, read-only data can be distributed into the cloud. This could be catalogue data like categories, articles or the search index.

8.2 Access to Backoffice Systems

Current shop systems are networking with various other systems, e.g. CMS, CRM, ERP, DBMS and others. For integration of these applications service-oriented approaches are often used. Most common are Web Services or RESTful Services. For consuming a service an application looks up a central service registry for fetching information about the service and invokes the service directly. For enhanced flexibility, services can publish the corresponding service descriptions on a P2P network. This will reduce maintenance and configuration issues, while adding availability to the lookup service. A drawback of a P2P-based lookup is a longer response time. In this scenario this seems to be acceptable as it is assumed that the configuration of backend systems does not change frequently and, thus, service lookups occur only on start-up and on configuration changes. Nevertheless, a server-based lookup-service implementing e.g. UDDI[4] could also be used.

[4] http://uddi.xml.org

8.3 Load Balancing

For providing load balancing and enhanced availability many systems use redundant web servers. A front-side web server redirects incoming clients to different server nodes. HTTP clustering is usually set up with static configuration, which decreases flexibility. The JXTA-based framework Shoal can operate across network borders and therefore is a good choice, as JXTA applies principles of self-organization[5]. If a system runs with a single superordinate balancer, simple presence protocols like XMPP[6] can be used for the exchange of presence information. Another possibility is to add a message queue to the balancer.

9 Conclusion and Outlook

The paper pointed out the benefits of a partial integration of cloud resources into an online shop for small- and medium-sized companies. The conflict between rising energy costs, fluctuating workloads and continuously high availability requirements will simply demand mechanisms that enable automated and dynamic scaling of such an application. Besides, largely underutilized servers cause unnecessary emissions.

Preliminary results of an analysis of a B2B system underline the problem clearly: Over a period of 23 days the mean CPU-utilization was 19.6 % with a maximum of 60 % and a minimum of 0.1 %, while the standard deviation amounts 30 %. This means, that the server was largely underutilized, while the deviation indicates a high fluctuation. This is supported by the fact that 80 % of the daily requests were received in twelve hours, while in the other twelve hours only the remaining 20 % were received. The steepest increase of 40 % points to the problem that the server utilization can change significantly within a few minutes. A conclusion of this study regarding dynamic scalability can be that components that are distributed on cloud resources require a short start-up time. This means that it is infeasible to load the full cache of e.g. catalogue data before being accessible. Ad-hoc queries in the already running instances could determine the most requested data items and prioritize those for caching.

For a deeper understanding of how the integration of cloud resources affects an organization, more studies on the following subjects need to be performed.

Cloud Computing: The TCO calculation in this paper is based on the Amazon Elastic Compute Cloud. To evaluate the economical benefits of the proposed strategy more providers have to be examined in consideration of services, pricing model and technical integration into applications. Comparisons of major providers can be found in Rad et al. (2009) and Hayes (2008). In the next step, we will exam-

[5] https://jxta.dev.java.net/ , https://shoal.dev.java.net/
[6] http://xmpp.org

ine the technical interfaces with which cloud providers expose the virtualized resources to customers to define a standardized way of managing those resources.

Automated Scaling: Just little is known about automated scaling. It is still not proven, how precise top-down capacity planning is in practice and how the data for e.g. forecasting business evolution can be determined and processed automatically – almost at real-time. Obviously this process should demand less computational resources than what can be saved by dynamic scaling. Thus, a bottom-up approach seems to be appealing: The cluster control would monitor all the nodes in the system and detects the situation in which an additional instance is needed or an instance can be shut down. However, as the demand of computational capability should be kept low, monitoring data cannot be transferred and analyzed in very short periods. This conflicts with the finding described above, that the utilization can rise significantly within minutes. It becomes obvious that neither of both approaches provides satisfying results. Therefore, influencing factors have to be studied in detail to combine the predictive approach with the reactive one.

Even if there are some obstacles remaining towards a fully automated mechanism, this paper shows the potential of dynamic scaling for small- and medium-sized companies who cannot balance hardware utilization. Fully outsourcing into the cloud is not attractive, as this creates dependencies. However, as the calculation in section 7 shows, a dynamically scaled application that integrates cloud resources results in significant economic and ecological savings and reduces the mentioned dependencies.

References

Alonso G, Casati F, Kuno H, Machiraju V (2004) Web services: concepts, architectures and applications. Springer, Berlin, Heidelberg.

Armbrust M, Fox A, Griffith R, Joseph AD, Katz R, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I, Zaharia M (2009) Above the Clouds: A Berkeley View of Cloud Computing. EECS Department, University of California, Berkeley. UCB/EECS-2009-28. http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html.

Brown P (2008) Implementing SOA: Total Architecture in Practice. Addison-Wesley.

Computerbase (2009) Riesiger Ansturm auf vergünstigtes Windows 7 (Update 2). http://www.computerbase.de/news/software/betriebssysteme/windows/windows 7/2009/juli/riesiger ansturm windows 7/, visited: 2009/07/23.

Chronicle M (2009) Das Windows 7 Pre-Order Drama! http://www.montie.de/2009/07/16/das-windows-7-pre-order-drama/, visited: 2009/07/23.

Hayes B (2008) Cloud Computing. CACM 51(7):9-11.

Hevner A, March S, Park J, Ram, S (2004) Design Science in Information Systems Research. MIS Quarterly 28(1):75-105.

Hevner AR (2007) A Three Cycle View of Design Science Research. Scandinavian Journal of Information Systems 19(2):87-92.

Illik, J (2002) Electronic Commerce. Oldenbourg Wissenschaftsverlag.

Kaufman C, Perlman R, Speciner M (2002) Network Security: Private Communication in a Public World. 2nd Edition. Prentice Hall PTR, New Jersey.

Laudon K, Traver C (2009) E - Commerce: Business. Technology. Society. 5th Edition. Pearson International Edition.

Menascé DA, Almeida VAF (2000) Scaling for E-Business. Prentice Hall PTR, New Jersey.

Menascé DA, Almeida VAF, Dowdy LW (2004) Perfomance by Design. Prentice Hall PTR, New Jersey.

Rad MP, Badashian AS, Meydanipour G, Delcheh MA, Alipour M and Afzali H (2009) A Survey of Cloud Platforms and Their Future. In: Gervasi O et al. (Eds.) Computational Science and Its Applications. Proceedings ICCSA 2009. LNCS 5592: 788-796. Springer, Berlin, Heidelberg.

Richter JP, Haller H, Schrey P (2005) Serviceorientierte Architektur. http://www.gi-ev.de/service/informatiklexikon.html. Informatiklexikon der Gesellschaft für Informatik.

Shapiro C, Varian H (1998) Information Rules: A Strategic Guide to the Network Economy. Harvard Business School Press, Boston.

Stahl E, Krabichler T, Breitschaft M, Wittmann G (2008) E-Commerce-Leitfaden. http://www.ecommerce-leitfaden.de, visited: 2008/05/20.

TecChannel (2009) Cloud Computing schürt Angst vor Kontrollverlust. http://www.tecchannel.de/test technik/news/1993816/sicherheitsbedenken bei cloud computing/, visited: 2009/07/23.

Turban E, Lee JK, King D, McKay J, Marshall P (2008) Electronic Commerce – A Managerial Perspective. Prentice Hall.

Wannewetsch H (2007) Integrierte Materialwirtschaft und Logistik. Beschaffung, Logistik, Materialwirtschaft und Produktion. 3. Auflg. Springer, Berlin, Heidelberg.

Zhang Q, Riska A, Riedel E, Smirni E (2004) Bottlenecks and their Implications in E-commerce Systems. In: Chi C-H, Steen M van, Wills C (Eds.) Web Content Caching and Distribution. Proceedings WCW 2004. LNCS 3293: 273-283. Springer, Berlin, Heidelberg.