Masaryk University
Faculty of Informatics

Software Architecture Optimization in Cloud Applications

Ph.D. Thesis Proposal

David Ge̊švindr

Supervisor: doc. RNDr. Tomáš Pitner, Ph.D.

Brno, January 2016

.....
supervisor

Declaration

Hereby I declare, that this paper is my original authorial work, which I have worked out by my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

David Gešvindr

Contents

1 Introduction

Cloud computing is rapidly gaining momentum and is significantly influencing the entire IT industry. Cloud computing defines new ways in which we interact with computers, run our applications and work with our data. Ten years ago, most of our work with computers was limited to desktop computers and the only reasonable form of mobile computing were notebooks. We used classical desktop applications, we stored all of our data locally on the devices we owned and fully controlled. Corporate productivity was supported by local IT departments hosting its on-premise services.

Over the last 10 years, the paradigm of our work with computer has significantly changed. Because of broadly available Internet connectivity, we became more productive, getting things done independently of where we actually are, seamlessly transitioning between high variety of devices. The devices give us access to our data which are not stored locally on the devices but rather managed by the applications or services which are operated in remote data centers using principles of the cloud computing with benefits of high operation efficiency and scalability.

Cloud computing together with mobile-computing devices and internet of things creates an environment which will result in an era of ubiquitous computing, where computers will be present anywhere at any time, completely surrounding us.

Cloud computing plays in this not-so-distant future a key role, as a connecting element providing on demand network access to a shared pool of computing resources, that can be rapidly provisioned and released with minimal management effort or service provider interaction. Thanks to these attributes, cloud computing attracts growing attention within both industry and academia as an alternative model of software system operation, associated with high system availability, scalability and performance.

However, the potential of cloud computing to optimize these quality characteristics is rarely taken to its fullest due to yet unclear dependencies of these qualities on proper software architecture design of the cloud-based systems.

Software architecture design is considered to be one of the key

phases of a software project, which significantly influences the overall quality of the software solution, independently whether the application is intended for the on-premise environment or for the cloud. For each new software paradigm it is necessary to explore and evaluate various architectures and continuously collect gained knowledge and transform it into a set of reusable architectural tactics and design patterns which lead to a design of applications with known and predictable behavior and software quality.

In case of the PaaS cloud, we have identified that such a design guidance is missing – despite the fact that design process of web applications, information systems or line-of-business applications operated in on-premise environment is well supported, we have not found in literature the same design support for PaaS cloud applications. By the same level of support we mean a set of design practices, architectural tactics and design patterns which were evaluated and proven suitable for given PaaS cloud environment with a well known impact on quality of the application.

Because of significant difference between the on-premise environment and the PaaS cloud, which is in detail explained and discussed in [22], we claim that an emergence of new design patterns and architectural tactics is necessary as the currently used practices may become anti-patterns in the PaaS cloud and vice versa tactics not efficient in the on-premise environment may be proven beneficial in the design of the PaaS cloud application to obtain required quality attributes.

The rest of this proposal is organized as follows: In Chapter 2 we introduce basic preliminaries and discusses existing approaches and related work. Chapter 3 propose the aims of this research and its outcomes. In Chapter 4 we present our achieved results.

2 State of the Art

2.1 Cloud Computing

To understand our research and its contributions it is necessary for the reader to be familiar with the cloud computing, therefore we introduce in this section important definitions and principles related to the cloud computing.

Definition 1. ***Cloud computing*** is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [30].

An integral part of the definition in [30] are also the essential characteristics of the cloud we consider to be very important as they represent a part of reasons how cloud differs from the on-premise environment:

- *On-demand self service* – A consumer can unilaterally provision computing capabilities as needed automatically without requiring human interaction with each service provider.
- *Broad network access* – Capabilities are available over network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms.
- *Resource pooling* – The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand.
- *Rapid elasticity* – Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.

- *Measured service* – Consumer is billed based on the measured resource usage.

As opposite to the cloud computing model, the on-premise computing model is defined in [25] as:

Definition 2. ***On-premise model*** – Is such a computing model when enterprises control their infrastructure and platforms at any time.

The definition of cloud computing defines three service models, which directly influence the level of control we have over the environment:

- *Software as a Service (SaaS)* – The capability provided to the user is in a form of hosted application which is fully operated by the cloud provider and the user has no control over the environment except settings exposed by the application. Good examples are *Office 365* [7], *Gmail* [5] – both being groupware applications provided as a service.

- *Platform as a Service (PaaS)* – The capability is provided in a form of a developer platform capable of running cloud applications without control of the underlying hardware infrastructure and the operating system. The user can control the deployed applications and configure settings of the application hosting environment. Notable representatives of this model are *Google App Engine* [6], *Azure App Service* [3].

- *Infrastructure as a Service (IaaS)* – The capability is provided to the user mostly in a form of a virtual server. Notable representatives of this service model are *Amazon Elastic Compute Cloud* [2], *Azure Virtual Machines*[4], *Rackspace* [8].

Cloud computing has also four defined deployment models: *Private cloud*, *Community cloud*, *Public cloud*, *Hybrid cloud*, depending on for whom the cloud infrastructure is provisioned. Despite the popularity of the public cloud where customers benefit from zero initial investment to hardware infrastructure [9], multiple companies and organizations are transitioning from their virtualized infrastructure towards private clouds [32]. They merge their compute resources into

a single pool of shared virtualized resources where individual company departments can host their virtual serves and services with advantages of the cloud, including rapid elasticity, so that they can scale the service depending on the load.

The cloud computing paradigm has a set of common properties as the grid computing [16, 33, 14], but the cloud computing is considered to be an evolution of the grid computing. From an application development perspective, despite some similarities, grid applications are generally more focused on high-performance batch computations without user interactivity. Grid applications are required to be designed specially for the given type of the grid. In the cloud we can operate the same set of applications as in an on-premise environment, which makes transition to the cloud appear trivial, therefore architects can overlook some differences which can negatively impact quality of the operated application.

Although the SaaS model can also be interesting to the software architect when searching for third-party components which can be integrated in their solution, quality and performance challenges are rather associated with IaaS and PaaS models, which give the architect more freedom in the design of the cloud application. Moreover, running IaaS virtual machines in the cloud environment is similar to running virtual machines in an on-premise environment. Therefore, PaaS cloud service model appears to be the most interesting from the quality-concerns perspective due to the lack of understanding of the built-in scalability and high-availability services, which are currently the key reason for many software architects to move their software applications to the cloud. These services include web hosting, storage and application services, which come with built-in high availability and easy vertical or horizontal scalability achieved through a complex service architecture [13] that includes multiple virtual hosting servers and complex network topology, including load balancers and built-in monitoring.

We are not denying the fact, that the same service can be build from the scratch in an IaaS virtual machines, but due to its complex deployment and configuration, the deployment and operation effort and costs will be higher that in case of PaaS. There are PaaS services which cannot be deployed in the IaaS because they were developed proprietary by the cloud operator or require acquisition of expensive

license. Next advantage of PaaS if zero-effort configuration, because even complex changes in PaaS service configuration are easily done via management API of the cloud provider, which in case of IaaS service would require more effort.

Thanks to these and another benefits, including the straightforward configuration APIs provided by the cloud provider, PaaS model is becoming the choice number one for many software architects who are considering to employ cloud computing in their architecture design [1].

2.2 Software architecture design

Because our work focuses on software architecture optimization, we in this section introduce the basic concepts of software architectures and then we focus on current state-of-the-art support for the design of cloud applications.

There are multiple very similar definitions of the software architecture [31, 28]:

Definition 3. A software architecture is a description of how a software system is organized. Properties of a system such as a performance, security, and availability are influenced by the architecture used [31].

Software architecture design is considered to be one of the most important activities in a software engineering project [11]. Software architecture design significantly influences overall quality of the application.

One of the main design challenges is defined as follows [24]: 'Software architect must make design decisions early in project lifecycle. Many of these decisions are difficult, if not impossible, to validate and test until parts of the system are actually built. Due to the difficulty of validating early design decisions, architects sensibly rely on tried and tested approaches for solving certain classes of problems. This is one of the great values of architectural patterns. They enable architects to reduce risk by leveraging successful designs with known engineering attributes'

Definition 4. Architectural patterns are a means of reusing knowledge about generic system architectures. They describe the architecture, explain when it may be used, and discuss its advantages and disadvantages [31].

Design patterns and tastics are differentiated in [10]. Architecture design of the application has a significant effect on its overall software quality.

With a set of well-described patterns and tactics for given environment, design methodologies can be applied, for instance *Attribute-Driven Design* [35], to design software architecture which meets given quality criteria [29].

There are multiple sources available [12, 19, 18, 17, 26] which provide comprehensive lists of proven architectural patterns and tactics but their validity for the cloud environment is not discussed.

Developers and architects are facing performance and quality challenges when using cloud services. We have identified the most intricate differences from an on-premise environment that every software architect designing a high-performance cloud application should be aware of [22].

Broad offer of platform services. Software architects tend to use equivalents of on-premise services available in the could, therefore they can miss services offered in the PaaS cloud with better performance characteristics. Frequent dependency of applications on the relational database is a common practice in the on-premise environment but in the cloud, scalability of the relational database is limited and high-performance PaaS database is a subject of high operation costs, therefore it is necessary to consider using an alternative service for storing application data which can result in use of NoSQL database service with high throughput and fraction of the operation costs. This transition needs to be supported by proper design guidance, which is currently missing.

Multi-tenant environment. In an on-premise environment we have distinct control over an allocation of resources for our applications, therefore we can plan in advance how applications can be co-deployed to share the same hardware resources without negatively influencing

each other. In the PaaS cloud it is a very common practice of cloud operators that applications of difference customers share the same infrastructure or at least some portion of it to increase an overall utilization of resources in the data center. This is manifested by a presence of noisy neighbor effect, which leads to a less predictable application performance. In our latest performance benchmarks we observed up to 30% variations in the throughput of the PaaS cloud application.

Cross-service communication latency. In an on-premise environment we have mostly precise control of network topology and we tend to deploy different components of the service collocated, to gain low network latency and high throughput. In the PaaS cloud we do not have this level of control over the infrastructure, therefore components of our service can be scattered over the entire data center, which leads to communication latency that in case of frequent component interaction increases the overall response time of the service.

Not all cloud services are designed to scale. It is important to precisely understand performance characteristic of the cloud service before it is utilized as a part of the cloud application as there are both platform services first with almost unlimited scalability when used as it is intended to, second services with limited scalability. The challenge is to distinguish those two groups of services, as their documentation typically does not include their benchmarks.

Pay-as-you-go billing model. Cloud providers offer high variety of cloud services with different billing models. We consider very important to analyze all costs related to operation of the platform service before the service is incorporated into an architecture of the application.

Undocumented behavior of the public cloud. The most intricate issues with the public cloud, not well documented by public cloud operators is the inconsistency in a service performance and transient faults. The former is caused by multitenancy of cloud platform services. The latter is caused by hardware and service outages in this hyper-scale environment, when commodity hardware is used and

high redundancy is achieved via a deployment of redundant computation and storage resources. Transient faults are results of forced failover operations which appear on a daily basis and is necessary to apply architectural tactics to avoid their manifestation in the application.

Quality conflicts. Software architect designing the application have to be aware of all quality attributes and keep them in balance, which is a challenging task due to conflicts between some of the quality attributes.

Because of the differences between the on-premise and the cloud environment validity of existing tactics and patterns cannot be guaranteed. Therefore we decided to search for existing patterns and tactics, which were proven in the cloud environment in general and then especially for the PaaS cloud model.

New catalogs of patterns relevant for the cloud are emerging [34, 15, 27], which are however not yet supported with demonstrations on real world use cases that could reveal the hints for efficient employment and combination of these patterns. The most detailed source [27] provides patterns descrition on a sufficient level of detail for their implementation and also outlines advantages and disadvantages of each individual pattern. However even this source lacks the discussion on effective combination of the patterns and their relation to the cloud-application quality attributes.

When we focused directly on the PaaS cloud, we have not found any design tactic or pattern that would be rigorously evaluated to use for the PaaS cloud application design with a measurable impact on application's quality.

Current design patterns are a good starting point but they need to be rigorously evaluated to measure their effect on software quality of the PaaS cloud application.

3 Aims of the Thesis

The aim of the PhD thesis is the introduction of the set of rigorously proven architectural tactics with a clear measurable effect on the quality of the designed *Platform-as-a-Service* cloud application.

Based on our knowledge and analysis of current state of the art, design support for PaaS cloud applications is significantly limited, missing suitable design techniques, patterns and tools that would lead to a design of a software architecture of a PaaS cloud application with a predictable software quality.

3.1 Objectives

We would like to reach our research aim via the following main objectives: *Introduction of architectural tactics* and *Design and development of a rapid prototyping tool*.

3.1.1 Introduction of architectural tactics

Architectural tactics are a proven method commonly used in a field of software engineering to provide software architects with a design support that leads to a design of software architecture with predictable software quality.

Our objective is to rigorously examine the impact of current state-of-the-art architectural tactics and design patterns available for on-premise applications on PaaS cloud applications. Based on the measurable impacts we will compose a proven list of identified tactics, design patterns and their combinations advisable for the design of PaaS cloud applications.

3.1.2 Design and development of a rapid prototyping tool

We identified both positive and negative impact on multiple aspects of cloud application's quality including performance – throughput, response time, scalability; availability and both development and operation costs. Because of revealed quality conflicts in tactic's impacts, a single list of tactics is insufficient support for software architect

who is responsible for the design of a PaaS cloud application because it is beyond the capability of the software architects to consider all positive and negative effects of their design decisions and precisely predict and quantify overall application quality without appropriate modeling tools.

Therefore our research effort leads to the design and development of a rapid prototyping tool which will leverage both current modeling approaches and rapid elasticity of the cloud. This tool will allow an automated conversion of the application model built using current state-of-the-art techniques and industrial standards for modeling into a performance prototype which will be in a form of a cloud application. This generated cloud application will interact with real platform services as described in the model, working with sample data sets generated and deployed based on meta information from the model and will be automatically published to the production PaaS cloud using provider's management API. On top of the deployed application a series of benchmarks will be executed to evaluate behavior of the application with a given software architecture in a real production environment and due to a cloud elasticity and measured service feature, it is feasible to quickly evaluate also large scale deployments with low effort and reasonable operation costs. The main features of the proposed tool are:

- **Use in an early development stage** – This tool will work as a decision support tool for a software architect since very early development stages of the project so that the architect can model and evaluate multiple possible architectures and decide which one will be used based on measured data. Alternative usage is in software engineering courses, so that students can experiment with various cloud architectures without the need to develop the application.

- **Low effort benchmarks** – Currently internal benchmarks during an early development of the application require implementation of the evaluated pattern in a form of application code, prepopulation of utilized data sources with sample data, use of tool generating the load, use of performance profiler and analysis of measured performance data. Our tool will simplify this

process as only the creation of the model is required, and measured performance data will be visualized back to the model, highlighting possible problems – overloaded components, components mostly affecting the response time, etc.

- **Collection of quality metrics relevant for architecture optimization** – Generated application will be orchestrated to collect a set of internal performance metrics valuable for identification of throughput and response times of individual operations and related PaaS platform services.

- **Library of patterns and tactics** – Library of our PaaS related software engineering tactics will be part of the tool, so that the user can easily apply this tactic in form of a snippet containing linked preconfigured operations.

- **Extensibility** – The tool will be designed in a modular way with an extensibility support allowing addition of new modeling languages, operations in the cloud environment and additional cloud providers.

As this tool will for the performance analysis use real cloud application deployed to the target hosting environment, it has the following advantages over current simulators, which simulates execution of the application:

- **Realistic results** – Because the application is executed in the target cloud environment, measured results are based on realistic interactions of the application with invoked platform services. I case of simulation tools, the environment and its behavior is only approximated with a certain degree of the veracity.

- **Current version of the cloud environment** – Evaluations of the modeled application are always executed in the current version of the cloud environment. Cloud providers continuously improve their PaaS cloud environment by application of multiple service updates, which mostly address bug fixes, security vulnerabilities or performance issues. To provide the same actuality of results using simulation techniques, it would be necessary

to do automated and costly benchmarks of all cloud platform services on periodical (near daily) basis to keep the simulation environment up-to-date.

We are also aware of following disadvantages and threads to validity of results when using the proposed tool:

- **Approximation of the operations** – Despite the fact, that operations available to be used in the model of the application are executed in the production cloud environment, they process only sample data of the similar range as expected to be in the production application.
- **Unrepeatable results** – Results describing behavior of the application are in the multi-tenant could environment impacted by the current state of hosting environment, which is beyond our control. Due to this reason, multiple runs of application's performance tests will result in slightly different, but realistic results.
- **Operation costs** – Because the evaluated cloud application will be published to the public cloud operated by the cloud service provider, service charges will occur depending on a variety of used platform services, their scale and generated load. Billing intervals vary service by service from minutes to days, which is information necessary for efficient reuse of allocated resources to minimize related costs of multiple evaluation runs.

3.2 Evaluation

Our findings related to software architecture tactics and patterns will be evaluated by application of these tactics on multiple cloud projects, where correctness of our findings will be confirmed by a set of measurements to analyze tactic impacts on quality attributes of the implemented application. Measured tactic impact will be rigorously compared with an expected impact.

The prototyping tool will be continuously validated during its development together with each new iteration in its development. As a

technique for its validation we propose comparison of obtained results from the prototyping tool, where multiple existing PaaS cloud applications will be modeled with results from a performance analysis of existing applications.

3.3 Expected Outputs Summary

- Publicly available catalog of rigorously proven design tactics and patterns with a measurable impact on PaaS cloud applications, including recommendations for their effective combination.
- Publicly available rapid prototyping tool for modeling and evaluation of PaaS cloud applications.
- Publications of theoretical results and practical evaluations of tactics.

3.4 Schedule

Spring 2016

- Thesis proposal submission (January 2016).
- Submission of paper describing architectural tactics for the design of PaaS cloud applications.
- Identification of new architectural tactics and their evaluation on cloud projects.
- Analysis and design of the prototyping tool.
- Implementation of the monitoring component which will collect performance metrics related to the quality of the cloud application in production environment.

Fall 2016

- Prototype implementation of the prototyping tool and its initial evaluation.

- Paper submission – Quality monitoring of PaaS cloud application.

Spring 2017

- Extension of the prototyping tool and its continuous validation.
- Paper submission – PaaS cloud application prototyping.

Fall 2017

- Doctoral thesis preparation.
- Doctoral thesis submission.
- Final version release of the prototyping tool.
- Final version of publicly available catalog of rigorously proven design tactics.

4 Achieved Results

In our research we focused on a deeper understanding of the PaaS cloud environment [20] and its relation to the software quality which was not comprehensively described in the related literature. As an outcome of this research we summarized and published performance challenges in PaaS cloud application design. Afterwards we contributed to the process of software architecture design of PaaS cloud applications by finding suitable software architecture design patterns and tactics for the design of efficient PaaS cloud applications.

Some of our contributions are also in the domain of Smart Grid, which is closely related to the cloud computing domain as both domains belongs to an ubiquitious computing. We applied our knowledge of quality driven architecture design from the field of cloud computing to help to identify research challenges towards Smart Grid quality by design. In the domain of Smart Grid design we also published our experience with an implementation of a Smart Grid Simulator. Experience and knowledge we have gained with simulations will be incorporated in a design of the rapid prototyping tool described as an outcome of this thesis.

4.1 Challenges in PaaS cloud application design

One of the key reasons why software architects are facing performance challenges when designing and developing PaaS cloud applications is that they tend to opt for services analogous to those they know from the fully controlled on-premise environment. They expect that these services hosted in the PaaS cloud will have the same characteristics and behavior as in the on-premise environment. By ignoring the key differences between the on-premise environment and PaaS cloud, they introduce to the applications a full range of performance bottlenecks and scalability problems.

We have identified the most intricate differences from an on-premise environment that every software architect designing a high-performance cloud application should be aware of [22].

In our research we have not only focused on identification of design challenges, but we have also analyzed multiple PaaS cloud ap-

plications and identified a set of common and frequent bad design practices that may be of a common use in the on-premise applications. As an outcome we have created a summary of useful practices for the design of PaaS cloud applications. We have structured our results based on impacted quality attribute.

We published our original results as a journal paper [22], which is attached to this thesis proposal.

4.2 Architectural tactics for the design of efficient PaaS cloud applications

As a part of our ongoing research we have reviewed existing design patterns and architectural tactics and analyzed their impact on the PaaS cloud applications. We have currently identified 6 architectural tactics with a significant measurable effect on a software quality of the PaaS cloud applications:

- *Materialized View* – Materialized View is an architectural tactic, which can be used to effectively combine multiple data storage services in the PaaS cloud. As already stated, relational databases does not scale well therefore we can use this tactic to create a prepopulated data views in a highly scalable NoSQL database service with low operation costs. We use the relational database as a primary data storage due to its integrity enforcement, transaction support and rich querying capabilities but we offload majority of the load to the NoSQL database. Measurements obtained from application of this tactic which show significant impact on throughput and scalability of the REST service hosted in Microsoft Azure App Service are depicted in *Figure 4.1*. It is important to be aware of limitations of this tactic as it is strongly advisable to combine it with *CQRS pattern* and *Asynchronous messaging tactic*.

- *Sharding Pattern* – Sharding Pattern divides stored data into a set of multiple horizontal partitions (shards) based on a partition key (sharding key). This tactic can be used to distribute data across multiple storage servers in the PaaS cloud, which leads

to significant increase of scalability of the storage if suitable partition key is used.

- *Static Content Hosting Tactic* – It is a common practice, that web and cloud applications use application servers to host a static content. In case of high-throughput PaaS cloud service it is advisable to use a dedicated storage service for hosting of the static content which offloads the application server which is a more expensive resource.

- *Command and Query Responsibility Segregation* – This design pattern is well described in literature, but we have identified that it significantly complements the materialized view tactic, as it separates in the application's architecture the read model and the write model. We take advantage of this architecture by altering update commands to update both relational database and materialized views in NoSQL database. It also allows us to serialize update commands and take advantage of the **Asynchronous messaging** tactic and increase throughput of update operations.

- *Asynchronous messaging tactic* – This tactic decouples producers and consumers of messages, in our case serialized commands. We take advantage of this tactic to efficiently scale compute and resource intensive operations, by instead of running the operation on the application server with a risk of client timeout, the application server schedules this operation as a serialized command to a scalable queue and confirms the client scheduling of the operation (in many cases client does not even need this information). A worker process loads commands from the queue and executes them. Throughput advantage of using this tactic for insertions of records which update materialized views is depicted in *Figure 4.2*. This can by extended by a *Competing Consumers tactic*.

- *Competing Consumers* – This tactic allows multiple consumers loading serialized commands from the queue which increases scalability of the *Asynchronous messaging tactic* but at the same time it brings new challenges that we needed to resolve.

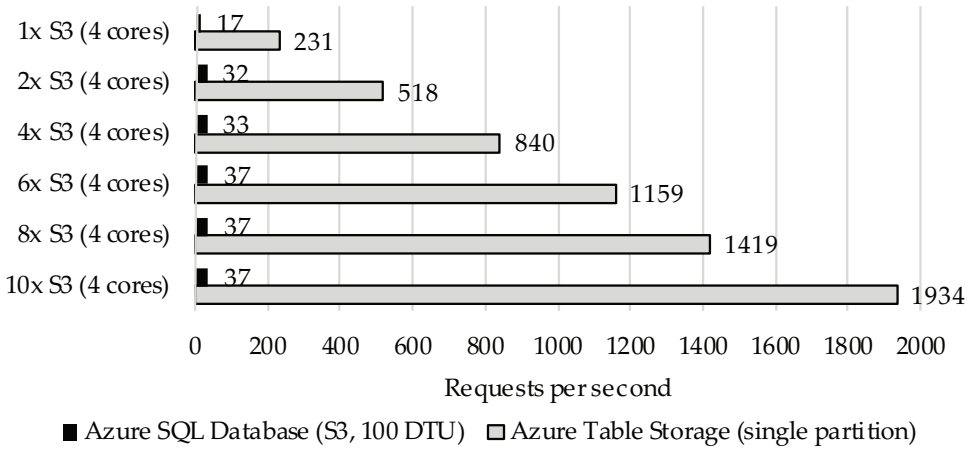


Figure 4.1: Comparison of the REST API read throughput with and without application of materialized view tactic

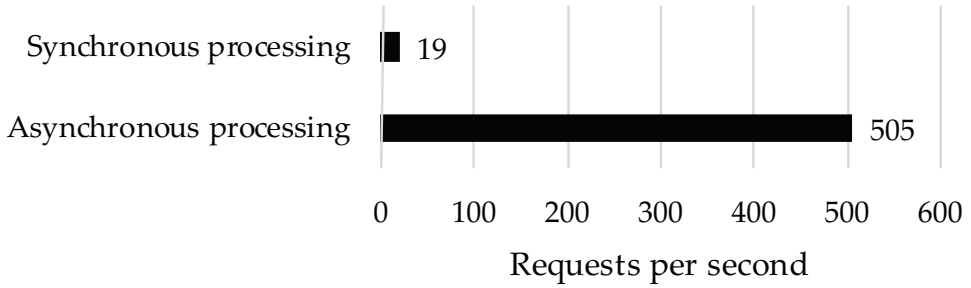


Figure 4.2: Comparison of the REST API write throughput with and without application of asynchronous messaging tactic

Our finding in this research area are well summarized in a conference paper 'Architectural Tactics for the Design of Efficient PaaS Cloud Applications' which is awaiting approval as it was submitted to the Working IEEE/IFIP Conference on Software Architecture with expected notification date in February 2016. Because the paper is completed and contains our original work it is attached to this thesis proposal.

4.3 Smart Grid quality by design

We applied our knowledge of software architecture design to the domain of Smart Grid, where we identified multiple research challenges towards Smart Grid quality by design, which we have published in [23].

We also published an experience report summarizing our results with Smart Grid Simulator design [21].

Both papers presenting our original results are attached to this thesis proposal.

Bibliography

[1] Paas market to reach $14 billion by 2017, idc says. http://www.infoworld.com/article/2609060/paas/paas-market-to-reach--14-billion-by-2017--idc-says.html, 2013. Accessed: 2015-01-24.

[2] Amazon elastic compute cloud. https://aws.amazon.com/ec2/, 2016. Accessed: 2015-01-20.

[3] Azure app service. https://azure.microsoft.com/en-us/services/app-service/, 2016. Accessed: 2015-01-20.

[4] Azure virtual machines. https://azure.microsoft.com/en-us/services/virtual-machines, 2016. Accessed: 2015-01-20.

[5] Gmail. https://www.gmail.com, 2016. Accessed: 2015-01-20.

[6] Google app engine. https://cloud.google.com/appengine/, 2016. Accessed: 2015-01-20.

[7] Office 365 for business. https://products.office.com/en-US/business, 2016. Accessed: 2016-01-20.

[8] Rackspace. https://www.rackspace.com/, 2016. Accessed: 2015-01-20.

[9] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, April 2010.

[10] Felix Bachmann, Len Bass, and Robert Nord. Understanding architectural patterns in terms of tactics and models. http://www.sei.cmu.edu/library/abstracts/news-at-sei/architect200708.cfm, note = Accessed: 2015-01-24, year=2007.

[11] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley, 2003.

[12] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-oriented Software Architecture: A System of Patterns*. John Wiley & Sons, Inc., New York, NY, USA, 1996.

[13] Brad Calder et al. Windows azure storage: A highly available cloud storage service with strong consistency. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 143–157. ACM, 2011.

[14] T. Dillon, Chen Wu, and E. Chang. Cloud computing: Issues and challenges. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pages 27–33, April 2010.

[15] Thomas Erl, Ricardo Puttini, and Zaigham Mahmood. *Cloud Computing: Concepts, Technology & Architecture*. Prentice Hall Press, Upper Saddle River, NJ, USA, 1st edition, 2013.

[16] I. Foster, Yong Zhao, I. Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE '08*, pages 1–10, Nov 2008.

[17] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[18] Elisabeth Freeman, Eric Freeman, Bert Bates, and Kathy Sierra. *Head First Design Patterns*. O' Reilly & Associates, Inc., 2004.

[19] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[20] David Gešvindr. Availability and scalability of web services hosted in windows azure. In *Proceedings of 10th Summer School of Applied Informatics. Brno*, pages 72–83, 2013.

[21] David Ešvindr and Barbora Bühnová. Smart grid simulator – an experience report. In *Proceedings of 12th Summer School of Applied Informatics, 14 pages. Brno*, 2015.

[22] David Ešvindr and Barbora Bühnová. Performance challenges, current bad practices, and hints in paas cloud application design. *ACM SIGMETRICS Performance Evaluation Review*, 2016.

[23] David Ešvindr, Barbora Bühnová, and Jan Rosecký. Overview of research challenges towards smart grid quality by design. In *Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on*, pages 1497–1504, Sept 2014.

[24] Ian Gorton. *Essential software architecture*. Springer Science & Business Media, 2006.

[25] P. Hofmann and D. Woods. Cloud computing: The limits of public clouds for business applications. *Internet Computing, IEEE*, 14(6):90–93, Nov 2010.

[26] Gregor Hohpe and Bobby Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[27] Alex Homer, John Sharp, Larry Brader, Masashi Narumoto, and Trent Swanson. *Cloud Design Patterns: Prescriptive Architecture Guidance for Cloud Applications*. Microsoft patterns & practices, 2014.

[28] ISO/IEC/(IEEE). ISO/IEC 42010 (IEEE Std) 1471-2000 : Systems and Software engineering - Recomended practice for architectural description of software-intensive systems, 07 2007.

[29] Suntae Kim, Dae-Kyoo Kim, Lunjin Lu, and Sooyong Park. Quality-driven architecture development using architectural tactics. *Journal of Systems and Software*, 82(8):1211 – 1231, 2009.

[30] Peter Mell and Tim Grance. The nist definition of cloud computing. 2011.

[31] Ian Sommerville. *Software Engineering*. Addison-Wesley Publishing Company, USA, 9th edition, 2010.

[32] B. Sotomayor, Ruben S. Montero, I.M. Llorente, and I. Foster. Virtual infrastructure management in private and hybrid clouds. *Internet Computing, IEEE*, 13(5):14–22, Sept 2009.

[33] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: Towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, December 2008.

[34] Bill Wilder. *Cloud Architecture Patterns*. O'Reilly Media, 1st edition, 2012.

[35] Rob Wojcik, Felix Bachmann, Len Bass, Paul Clements, Paulo Merson, Robert Nord, and William Wood. Attribute-driven design (add), version 2.0. Technical Report CMU/SEI-2006-TR-023, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2006.

A List of Author's Results

The ongoing results of our research were publicly presented on the following conferences or published in proceedings and journals:

A.1 Proceedings Publications

2015

- GEŠVINDR, David, Barbora BÜHNOVÁ. *Smart Grid Simulator – An Experience Report*. In Proceedings of 12th Summer School of Applied Informatics, 14 pages. Brno, 2015.

  Contribution: 90%. I implemented the simulation environment and wrote the text.

2014

- GEŠVINDR, David, Barbora BÜHNOVÁ and Jan ROSECKÝ. *Overview of Research Challenges towards Smart Grid Quality by Design.* In Proceedings of the 2014 Federated Conference on Computer Science and Information Systems, pp. 1497–1504. IEEE, 2014.

  Indexed in Web of Science.

  Contribution: 70%. I collected the challenges relevant to the Smart Grid quality by design. I also wrote major part of the text.

2013

- GEŠVINDR, David. *Availability and scalability of web services hosted in Windows Azure.* In Proceedings of 10th Summer School of Applied Informatics, pp. 72–83. Brno, 2013.

  Contribution: 100%. I am the only author of the paper.

A.2 Journal Publications

2016

- GEŠVINDR, David, Barbora BÜHNOVÁ. *Performance Challenges, Current Bad Practices, and Hints in PaaS Cloud Application Design.* In ACM SIGMETRICS Performance Evaluation Review, 10 pages. To appear in ACM, 2016. ISSN 0163-5999.

  Indexed in Scopus. To appear.

  Contribution: 70%. I collected the challenges relevant to PaaS cloud application design and analyzed them based on my experience with cloud projects. I wrote most of the text.

A.3 Publications in an approval process

2016

- GEŠVINDR, David, Barbora BÜHNOVÁ. *Architectural Tactics for the Design of Efficient PaaS Cloud Applications*. Submitted to Working IEEE/IFIP Conference on Software Architecture (WICSA), 10 pages. IEEE, 2016.

  CORE Rank A.

  Contribution: 70%. I identified relevant architectural tactics, described their impact and conducted case-studies. I wrote most of the text.

A.4 Presentations

2015

- GEŠVINDR, David. *Software architecture design of a scalable cloud service.* PV179 – Selected Topics in .NET Technologies, Faculty of Informatics, Masaryk University, December 2015

- GEŠVINDR, David. *How we implemented small Facebook for elementary schools in Microsoft Azure.* TechTalk event, Microsoft Czech Republic, Prague, November 2015

- GEŠVINDR, David. *Challenges of quality management in cloud applications.* PV260 – Software Quality, Faculty of Informatics, Masaryk University, May 2015

- GEŠVINDR, David. *Azure SQL Database.* Azure Global Bootcamp conference, Brno, April 2015

- GEŠVINDR, David. *Software architecture design of a scalable cloud service.* Azure Global Bootcamp conference, Brno, April 2015

- GEŠVINDR, David. *Azure SQL Database.* Windows User Group event, Ostrava, February 2015

- GEŠVINDR, David. *Azure SQL Database.* Windows User Group event, Brno, January 2015

2014

- GEŠVINDR, David. *Azure SQL Database.* Windows User Group event, Praha, November 2014

- GEŠVINDR, David. *Software architecture design of a scalable cloud service.* TechEd 2014 professional conference, Prague, May 2014

- GEŠVINDR, David. *Software architecture design of web services hosted in the cloud* PV226 – Seminar LaSArIS, Faculty of Informatics, Masaryk University, April 2014

A.5 Teaching and Thesis supervision

I supervised five bachelor's theses and consulted one master's thesis. I reviewed three bachelor's theses and two master's theses.

- PV178 – Introduction to Development in C#/.NET (2014, 2015)

- PV179 – Selected Topics in .NET Technologies (2013, 2014, 2015)

- PV239 – Mobile Application Development (2014, 2015)

A.6 Awards

2015

- **Microsoft Most Valuable Professional Award** in Azure cloud expertise – Rewarded for an exceptional contribution to the professional cloud community in the Czech Republic

2013

- **Dean's Award for final thesis** – Diploma thesis topic: *Availability and scalability of web services hosted in Windows Azure*

A.7 Certifications

- **Microsoft Certified Solutions Developer: Windows Store Apps using C#**
  Since 11/2016
- **Microsoft Certified Solutions Expert: Data Platform**
  Since 03/2014
- **Microsoft Certified Solutions Associate: SQL Server 2012**
  Since 01/2014
- **Microsoft Certified Professional Developer: ASP.NET 3.5**
  Since 05/2010
- **Microsoft Certified IT Professional: Database Developer 2008**
  Since 05/2010
- **Microsoft Certified IT Professional: Business Intelligence Developer 2008**
  Since 01/2010
- **Microsoft Certified IT Professional: Database Administrator**
  Since 06/2009
- **Microsoft Certified Trainer**
  Since 06/2009

B Attached publications

1. GEŠVINDR, David, Barbora BÜHNOVÁ. *Smart Grid Simulator – An Experience Report*. In Proceedings of 12th Summer School of Applied Informatics, 14 pages. Brno, 2015.

2. GEŠVINDR, David, Barbora BÜHNOVÁ and Jan ROSECKÝ. *Overview of Research Challenges towards Smart Grid Quality by Design.* In Proceedings of the 2014 Federated Conference on Computer Science and Information Systems, pp. 1497–1504. IEEE, 2014.

3. GEŠVINDR, David. *Availability and scalability of web services hosted in Windows Azure.* In Proceedings of 10th Summer School of Applied Informatics, pp. 72–83. Brno, 2013.

4. GEŠVINDR, David, Barbora BÜHNOVÁ. *Performance Challenges, Current Bad Practices, and Hints in PaaS Cloud Application Design.* In ACM SIGMETRICS Performance Evaluation Review, 10 pages. To appear in ACM, 2016. ISSN 0163-5999.

5. GEŠVINDR, David, Barbora BÜHNOVÁ. *Architectural Tactics for the Design of Efficient PaaS Cloud Applications*. Submitted to Working IEEE/IFIP Conference on Software Architecture (WICSA), 10 pages. IEEE, 2016.

Smart Grid Simulator – An Experience Report

[1]David Gešvindr, [2]Barbora Bühnová

[1] Masaryk University, Faculty of Informatics
Botanická 68a, 602 00 Brno
gesvindr@mail.muni.cz
[2] Masaryk University, Faculty of Informatics
Botanická 68a, 602 00 Brno
buhnova@fi.muni.cz

Abstract
The deployment of a Smart Grid is becoming a strategic act for many countries, including the Czech Republic. However, the transition to the Smart Grid represents heavy investments into technology that was never before deployed in such a large scale. To minimize the investment risks, power distribution companies and governments are seeking new ways of Smart Grid modelling and simulation to evaluate and compare different Smart Grid design alternatives. In this paper, we share our experience with the design and implementation of a Smart Grid simulator, and namely its network communication module, in the Czech Republic.

Keywords
Smart Grid, network simulator, discrete event simulator, design validation, communication strategy, co-simulation

1 Introduction

A Smart Grid is a new generation of power distribution grid which actively utilizes information technologies for real time monitoring and control of the power grid. This approach opens new possibilities of the grid control and related demand-response optimization but also brings new challenges in the Smart Grid design. To prevent Smart Grid design flaws, new simulation frameworks are being developed to facilitate the prediction of Smart Grid properties under different utilization scenarios.

Goal of the paper: The goal of this paper is to share our experience with the design of a smart grid simulator, and more specifically its network simulation module optimized for large scale evaluation of a smart grid communication strategy, in the Czech Republic.

Paper structure: The paper is organized as follows: Section II introduces the reader to the benefits of future Smart Grid deployment and related problems that need to be solved; Section III overviews various simulation techniques and existing solutions for network simulation; Section IV describes our approach to Smart Grid simulator design, its architecture and details the design of the network simulation module; Section V summarizes the lessons learned; and Section VI concludes the paper.

2 Background

This section introduces a reader to the benefits of the future Smart Grid deployment and related problems that need to be solved.

2.1 From Power Grid to Smart Grid

The key principles of current power grids did not change since their inception in the industrial age in the early 19th century. Power grids in the Czech Republic and elsewhere were designed for a centralized power generation in a small number of large power plants located close to the natural resources and efficient transmission of generated electricity to the customers. To minimize transmit losses, the generated electricity is transformed using step-up transformers up to 400 kV and is transmitted using transmission network over large distances. In the target area the high voltage electricity is transformed down to 110 kV using step-down transformers which act as a gateway between the transmission and the distribution network. The distribution network then consists of power lines and large second level substations which transform electricity down to 22 kV or 35 kV. Third level substations are then located closely to urban areas and process the final transformation of electricity down to 230/400 V which is a voltage supplied to consumers. More detail about the power grid architecture in the Czech Republic can be found in [1].

In a Smart Grid, consumers are equipped with a smart meter device, which measures their electricity consumption and delivers these measurements to the closest collection point typically located at the third level substation (distribution transformer stations, DTS) which is equipped with a data concentrator. It is an additional device installed at the substation dedicated to a measurements collection and processing. This data concentrator then uploads aggregated measurements to the data center for global processing.

The Smart Grid is an attempt to solve major problems that occurs in power grids of the current generation – a complicated demand response optimization and a distributed energy generation. The former problem is caused by a difference between an off-peak and a peak load which requires complicated adjusting of the power generation capacity. The latter problem is caused by massive inclusion of photo-voltaic power plants into the power grid which leads to uncontrolled overflows of the energy during sunny days with a peak production.

The proper functionality of the Smart Grid is crucially dependent on the ability to transfer measurement data from consumers and control command to the consumers. The following sections describes more in a detail communication technologies used in the Smart Grid.

2.2 Communication in Smart Grid

Based on the Smart Grid topology mentioned in the previous sections there is a strong dependency of a correctly working Smart Grid on a reliable information exchange between smart meters, data concentrators and datacenter.

The following basic types of information are exchanged:

- consumption measurements from smart meters to data concentrators,
- aggregated consumption measurements and substation state information from a data concentrator to a data center,
- local power plant production measurements (eg. photovoltaics), and
- control commands to smart meters to switch between the high and low tariff.

There are multiple communication technologies applicable to these purposes with certain benefits and limitations. Specific technologies that are evaluated for the deployment in the Smart Grid in the Czech Republic are discussed in [1].

2.3 The purpose of Smart Grid simulation

Construction and deployment costs of a Smart Grid in a nation-wide scale demands a precise evaluation of all characteristics of a Smart Grid deployment to ensure its qualitative characteristics. Detailed

modelling of a Smart Grid is necessary for a pre-deployment evaluation but is not sufficient to evaluate key behavioral characteristics of a complex system, mostly of the ICT layer of the Smart Grid, which is a key dependence of a correctly working Smart Grid. To evaluate the behavior of various components in the system we need to run the model of the Smart Grid in a simulation environment. This approach allow us to study complex interactions between abstracted components of the system and evaluate various design solutions with low cost. After certain solutions are proven working, they can be deployed in small scale to observe other qualitative characteristics without inaccuracies brought by an abstraction during model creation.

The following list provides a subset of research questions from our previous work [1] which can be at least partially answered by Smart Grid simulation:

- What shall be the level of centralization within the Smart Grid?
- What communication channels and communication strategies are suitable for control channels? What are the requirements that control channels need to meet?
- How can we increase the reliability of the line? How can we get cost-effective line redundancy?
- Shall the substations and other grid components be organized in a hierarchical manner? If yes, how many levels should be used?
- What design strategies can be used to maximize the quality of the Smart Grid under the constraint of legacy infrastructure?
- What hardware infrastructure changes will imply the highest ratio of quality increase to implementation costs (i.e. quality/cost)?
- Are the current communication channels suitable for reliable control of customer appliances or real-time delivery of the price? How does the system behave in case of a broken communication line?

3 Simulation

This section briefly introduces a reader to various simulation techniques and details the Discrete Event Simulation. The last part of this section enlists selected network simulators and discusses limitations of their usage for Smart Grid simulation.

3.1 Introduction of various simulation techniques

Simulation techniques allow imitation of an operation of a real world system over time. There are multiple simulation classifications that vary based on simulated system characteristics. We discuss the distinction between continuous and discrete time simulation.

In a continuous simulation the modeled system is described with equations and our goal is to find a state in which the system is in an equilibrium with respect to these equations. The continuous simulation is frequently used for solving problems related to dynamic systems. In a continuous dynamic simulation the system is described with differential equations. Application of this approach is in a simulation of electric circuits, a chemical process modeling or a population modeling. Continuous simulation continuously tracks the system dynamics over time.

Discrete time simulation in contrast to the continuous simulation models the system in a sequence of events in time, therefore it is also called action based simulation. Each event marks a change in the system and it is assumed that no other change to the system occurs in the time between events.

An alternative approach is the process-based simulation, where each activity in the system corresponds to a running process in the system. Discrete events generated by the process cause other processes to sleep, wake up or update their state.

3.2 Discrete Event Simulation

A discrete event simulator is comprised of the following basic components: state, clock, events list and a random number generator.

The simulator maintains the state of the simulation environment and the way in which this state is stored depends fully on the implementation of the simulator. It can be hold in variables or in an inner state of complex objects if the simulator is object oriented. The use of object oriented paradigm has its benefits in a discrete event simulator design, because it naturally mimics the real-world objects being simulated, their states and interactions.

Discrete event simulator has to maintain a simulation time. The clock component is a very simple component that stores simulation time of the last or currently processed event and passes this information to an event list and an application code running the simulation logic.

The event list component is a storage for simulation events. During the initialization of the simulation run this event list is loaded with the initial simulation events that are scheduled for future processing. The simulation itself is an infinite loop with the following steps:

1. Get the oldest event in the event list and check if its timestamp is equal or greater than the simulation clock.
2. If the event list is empty or the stopping criteria are satisfied, the simulation is stopped.
3. Update the simulation clock with the timestamp of the processed event.
4. Execute application code that processes the event which may lead to changes in the state of the simulation environment and also to a scheduling of future actions in form of new events with timestamp set to current or future simulation time.
5. Store generated events in the event list

3.3 Limitations of existing network simulators

Since an evolution of network protocols, multiple network simulators were developed and are currently available for free or as paid products. Existing simulators offer wide variety of features and supported protocols. The majority of network simulators use discrete event simulation. Existing simulators focus primarily on the full support of all protocol features and a high precision of running simulations which leads to a high volume of generated and processed events during the simulation.

The following list presents some of the existing network simulators which are frequently used in an academic environment and offer free license for academic purposes:

- **ns-3** (network simulator 3) is a discrete event network simulator published as a free software with a rich functionality.
- **OMNeT++** (Objective Modular Network Testbed in C++) is a discrete event simulation environment. Its primary application area is the simulation of communication networks, but because of its generic and flexible architecture, it is successfully used in other areas like the simulation of complex IT systems, queueing networks or hardware architectures [2]. The **INET** Framework is considered the standard protocol model library of OMNeT++. INET contains models for the Internet, wired and wireless link layer protocols etc. [3]
- **NetSim** is a state-of-the-art network simulator for lab experimentation, R&D and defense applications [4].

The primary limitations we identified that block us from using these simulators are:

- **Integration complexity** – The studied simulators provide a level of extensibility that is insufficient for designing and implementing the entire Smart Grid simulator, whose intended functionality reaches far beyond network simulation. Even if autonomous simulators are available for different simulation tasks, the integration of the simulators is a very complex task. We have experienced this situation in cooperation with our industrial partner who already had an implemented Smart Grid modelling tool (called GridMind), which was however implemented in prolog and our analysis revealed that its integration with a C++ network simulator would be challenging.
- **Level of detail** – Existing network simulators focus on a detailed and exact network simulation across all network layers. This approach is necessary for use cases related to the analysis of network protocols. On the other hand, Smart Grid models do not necessarily contain all the information about Smart Grid components, namely at the physical, link and internet protocol layer, therefore the lower network layers are better to be approximated within Smart Grid simulation.
- **Computational complexity** – During the simulation, the existing network simulators generate high volume of simulation data, basically for each transmitted frame, which in combination with the previous point implies unnecessary load that could be avoided.

Because of the previously mentioned facts we decided to build a new simulation solution primarily focused on Smart Grid ICT simulation that provides high degree of modularity and extensibility.

4 Our approach

Our approach to smart grid simulation is based on designing highly modular and extendable simulator with a centralized storage for simulation events. We are overcoming most of the co-simulation complexity by using synchronous event processing across all modules involved in the simulation. The following sections describe the specific components of our simulator.

4.1 Simulator architecture

The proposed simulator architecture was designed to be modular and easily extendable. It is comprised of a central simulation core and a set of independent simulation modules, each simulating a different aspect of Smart Grid behavior. This paper discusses mainly the network simulation module.

4.1.1 Central timeline component

The core component of the proposed Smart Grid simulator is a central runtime with a shared storage maintaining simulation events used for running discrete event simulation. This component is responsible for storing high volume of simulation events while providing high event throughput between the core and simulation modules. During the simulation run this central queue, called the timeline, sends the oldest available event to modules capable of processing the event. After independent processing of an event is done by a simulator module, newly generated events are stored back in the queue under the condition that their timestamp is equal or greater than the processed event. This is a necessary condition to maintain correct time flow in the simulation, it basically blocks action scheduling into the past.

The second responsibility of this core component is the maintenance of active simulation modules. Each simulation module loaded into the system defines its communication interface in a form of an entry point accepting events in a module-specific format. A part of a module definition is a list of allowed input and output events. There are defined dependencies between input and output events that are used for validation of module's output and for later analysis and testing. Every module in the system can define a subscription to an event type that it is capable to process. The central core is then routing events from timeline to modules based on defined subscriptions. Because of heterogeneity in the used modules

and their various implementation technologies, there is sometimes a need to convert events between various formats accepted by simulation modules. Therefore event subscriptions can be associated with a converters translating event to a format accepted by a module running the event processing.

The core of the simulator is implemented in the Prolog programming language as an integral part of the GridMind modelling tool developed by our industrial partner Mycroft Mind. Our contribution is in the selection of simulation approach, design of simulator architecture and implementation of a network simulator module written in .NET. To connect the Prolog simulator core with the module written in .NET, we had to define the exact communication interface used for the transfer of simulation events from the central time line to the processing module. After processing the event, new events may be generated as a result. Therefore this communication interface is responsible for two-way communication between the timeline and the module.

4.1.2 Event subscription model

The timeline module holds a list of subscriptions made by simulation modules to receive certain types of events when they are loaded from the central simulation queue. This event routing architecture was designed to support our modular approach and isolation of various modules in the simulation. The event subscription model is substantial when defining connections between various simulation modules. A simulation module is working with its own set of simulation events sent to the timeline and after withdrawal sent back to the module, but when another module has to react to a certain event in the simulation, it needs to subscribe to an event of different module. For example:

1. Application protocol initiates data transfer between two end points in the Smart Grid. This is done via creation of a *Data Transfer Started* event which is routed to the Network Simulator module
2. The Network simulation module internally initiates the data transfer in its data structures which leads to the generation of data packets that need to be transferred over the line. Two types of events are queued in the central timeline – *Data Packet Sent* and *Data Packet Received*.
3. The Network simulator is subscribed to *Data Packet Sent* and *Data Packet Received* events and is handling their processing.
4. When low level communication is completed, the Network Simulator generates the *Data Transfer Finished* event. The Network simulator is not subscribed to this event, because it does not need to handle it, the transfer is finished internally, instead of that it notifies the other modules interested in this event – subscribed to it.
5. *Data Transfer Finished* event is handled by the application protocol simulator which continues in processing of the communication schema given by the used application protocol.

4.1.3 Heterogeneous module integration

The simulator architecture has to support growing portfolio of simulation modules being currently developed. The modules that are under development are focused on the simulation of IT infrastructure of a Smart Grid, not the power management characteristics, and includes the DTS behavior simulation and communication simulation.

The DTS simulator manages the instances of DTS devices and their lifecycle in the simulation environment. The DTS devices are instantiated by the GridMind model, which holds the entire model of the Smart Grid and configures various simulation modules with the goal to create instances of the simulated devices based on the parameters of the model.

The communication simulator does not work as a monolithic network simulator simulating all the layers of communication in the Smart Grid, instead the communication is simulated using multiple interconnected simulation modules. Each is designed for a specific protocol or communication layer.

The bottom of our network stack is the network simulation module described more in the following sections. The main purpose of this module is the provision of basic network services to application protocols. It manages network infrastructure and provides basic data transfer simulation.

On top of the network transfer simulation there are additional simulation modules simulating various application protocols used for the communication between DTS and the data center.

4.2 Network simulation module

Our network simulator is responsible for the simulation of lower part of the network stack – including physical layer, link layer, internet layer and transport layer. It runs as a standalone application communicating with the central timeline using its communication interface. The following sections describe the implementation process beginning with the requirement analysis until the testing and validation.

4.2.1 Requirements identification

Prior to the design of the network simulation module we had to identify the requirements emerging from different modules involved in the network simulation of the application protocols.

As the basic simulation scenario we identified the simulation of data transfer. Data transfer is parameterized by:

- used data line,
- used transport protocol,
- amount of transferred data.

The application protocol initiates data transfer and needs to be notified when the data transfer finishes so that it can continue in scheduling other actions.

As mentioned in Section 2.2 there are multiple communication technologies that need to be evaluated during Smart Grid design. Our goal is to find a simplified way of their simulation based on an abstraction from the physical layer. To abstract the physical layer of the network stack we combined a rich set of parameters describing this layer into three primary parameters of the simulated line – line speed, line latency and error rate. There are certain inaccuracies brought to the simulation by this simplification, but their impact can be omitted as a part of our simulator abstraction. To simulate special behavior of certain line types (eg. increased latency of the first packet after a period of GPRS line inactivity) we have to support intercepting *Packet Sent* and *Packet Received* event processing.

The most widely used transport protocols are the TCP and UDP. The former is frequently used to transfer data from a DTS to a datacenter. Despite the fact that the latter is used in a minority of transfer scenarios, the simulator has to support both transport protocols to be able to evaluate all required communication scenarios.

The transport layer of the network stack operates over data streams provided by the application layer and is not aware of a file or a request boundary. For the purpose of our simulator we designed very thin and simplified application layer that enables other modules to work with abstract data transfer of a given size instead of continuous streams of bytes.

4.2.2 Evaluation of various protocol features that need to be implemented

Network protocols, particularly transport protocols, offer a rich set of features which can be precisely simulated by existing network simulation solutions. The computational complexity steeply increases with the fidelity of the simulation, therefore we need to identify and simulate only the features of network protocols that significantly affect simulation precision. The rest of the protocol‘s functionality can be omitted in case that the depreciation of simulation fidelity does not affect the behavior of the simulated systems.

The second and equally important criterion for selection of protocol abstraction level is the level of detail of the Smart Grid model that does not contain exact definitions of all possible communication lines being used particularly when connection is routed through public internet lines. Therefore the line descriptions in the model are limited to be point-to-point and are parameterized at the most limiting line.

In the following section, we discuss the major protocol functionality and its impact on the simulation fidelity:

- **Link Layer –** This layer which is in the Internet Protocol Suite described as a combination of OSI physical layer and data link layer is in existing network simulators simulated by sending individual data frames. This approach has two main disadvantages for us: it requires a detailed network topology model and is computationally demanding. The former is a problem because of missing detailed network topology description in the Smart Grid model, the latter leads to high volume of simulation events, which we are trying to avoid. To overcome the previously mentioned disadvantages, we are simplifying this layer to be in our simulator represented by a parameterized network line of certain transfer speed, error rate and latency.
- **Internet Layer** – The internet layer is responsible for routing the packets through the public internet between various networks. The situation is the same as in the case of the link layer. It is out of our scope to simulate routing over the public internet. This layer can be simplified without overall impact on the data transfer duration. Based on the version of the used IP protocol we have to include appropriate header payload among transferred data.
- **Transport Layer** – We tried to simplify the transport layer, but our tests have shown that this layer, especially the TCP protocol and its behavior, cannot be omitted. The first TCP feature we had to implement was 3-way handshake and connection handling that on lines with higher latency and error rate has significant impact on the duration of relatively small data transfers which are typical for certain Smart Grid communication scenarios. We had to extend our model of the simulator to support opening and closing of the connection together with a validation of a connection presence between endpoints at specific ports. Then we had to support a send window and receive window mechanism together with congestion control principles and resend timers. The previously mentioned mechanisms can significantly decrease data throughput of a network line which can lead to communication problems in the smart grid, which we want to detect. Therefore the mentioned mechanisms cannot be omitted and need to be simulated.
- **Application Layer** – It is not a purpose of our simulator to simulate application protocols. They are handled by other modules, but we wanted to provide previously mentioned mechanisms abstracting from work with data streams at the transport layer and providing simple interface in a form of Data Transfer object and related events.

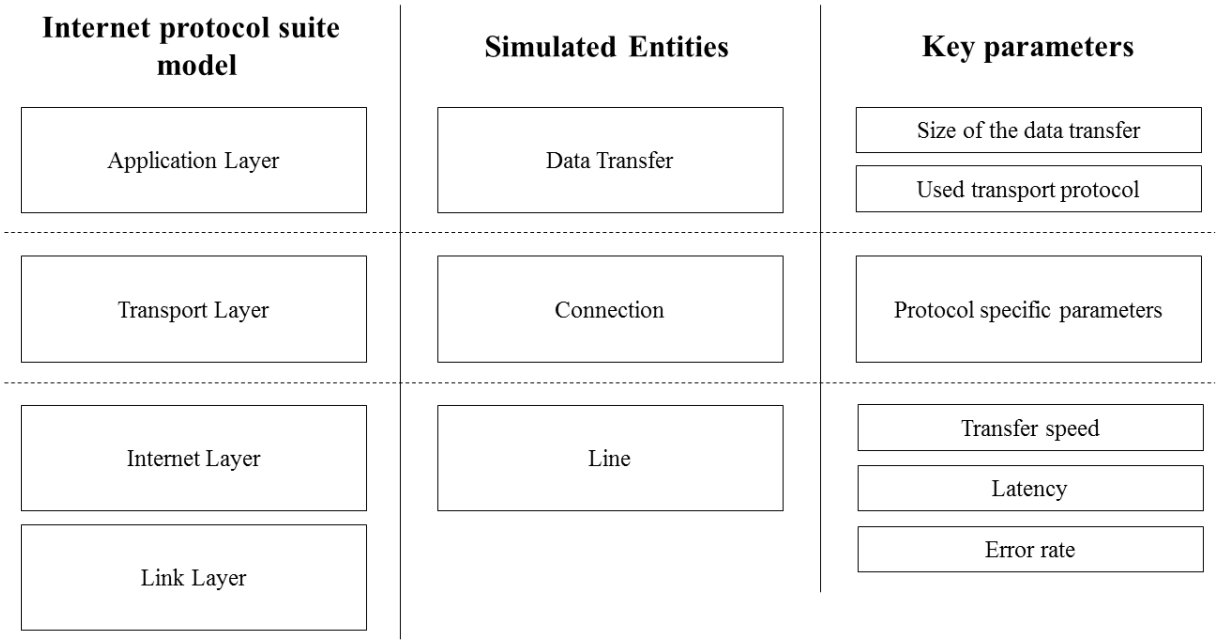


Figure 1: Mapping of Internet protocol suite model layers to simulated entities and their key parameters

To sum up our abstractions, we combined the lowest 2 levels of the network stack, implemented the transport layer in detail and created a thin stream abstraction layer on top of it to simplify application protocol implementation.

4.2.3 Iterative evaluation of accuracy level changes with increasing functionality being simulated

Our initial vision was to build and keep this network simulator as simple as possible. Because our aim is not to analyze detailed protocol behavior we do not need to simulate all of the protocol‘s functionality. Studies involving protocol behavior has already been conducted when new protocols evolved. Our focus is primarily on computation of data transfer duration under given network conditions.

We work with the following simplifications:

- **Point to point connections** – Our focus is primarily on data transfers between a DTS and a data center. There are multiple networks the communication is routed through but the overall performance of the short data transfer is mostly affected by the bottleneck – the worst performing line, typically mobile broadband used by DTS. Therefore we simulate only the line which is the bottleneck.

Design of this simulator has passed through 3 major iterations which are described in the following section:

1. **Flow variant** – Because of the focus only on data transfer duration over point-to-point line of a certain speed, we initially evaluated an idea, that the duration can be computed using the following equation:

$$data\ transfer\ duration = \frac{size\ of\ transferred\ data\ [B]}{allocated\ line\ speed\ [B/s]}$$

   In this iteration we have started the data transfer, checked the line speed currently available and computed in advance the possible end of the data transfer. The main point is that we generated only *Line Check* event instead of *Data Transfer Finished* event, because a line can change its

parameters or a new data transfer can be started while the original one is still running. This leads to a change of allocated line speed we initially distributed evenly among all data transfers sharing the same line. Consequently the expected ends of data transfers become delayed. We stood in front of a decision how to invalidate already queued ends of data transfers. One option was to remove event from the timeline signaling the end of data transfer without its processing which would increase complexity of this critical timeline component. The second option was to use previously mentioned *Line Check* event which signals possible end of data transfer on a given line. When *Line Check* event is processed, it validates how many bytes on running data transfers were already transferred and identifies data transfers that finished at the same moment. If there was a data transfer that has finished, it generates *Data Transfer Finished* event and also a new *Line Check* event for the next ending data transfer.

This approach is very effective in an environment meeting the following conditions:

- Simple transport protocol, suitable only for UDP
- Reliable line with low latency – there is no direct method how to simulate lost or damaged packets
- Low amount of line changes or parallel data transfers

Results of this approach significantly differ from reality when this approach is used for TCP or on an unreliable line with latency. Therefore we had to evaluate our second approach.

2. **Simplified packet variant** – After the identification of strengths and weaknesses of the flow approach, mostly the inability to accurately simulate TCP data transfers, we decided to evaluate a new approach where data are being divided into atomic units – packets. There is only one packet at a time being transferred over the network line. This division into packets moves our approach towards the way computer networks physically work. This change required us to design and implement new mechanisms for handling data transfers which are split into small chunks of data. If there are more data transfers over the same line, there is a round robin algorithm that serves justly all running transfers.

   The line does not send multiple packets simultaneously. The packet being currently sent utilizes full line capacity when queuing the *Packet Sent* event. Processing of this event involves getting the next packet from running data transfer using the round robin algorithm. When the last packet of the data transfer is sent, *Data Transfer Finished* event is queued into central timeline.

   The packet version of the simulator simplified handling of available line capacity changes. For long running data transfers the use of packets has led to an increased number of events in comparison to flow variant.

   Due to the simplifications at the transport protocol level this variant is not suitable under following conditions:

   - TCP transfers over lines with high latency and error rate – Frequent scenario in a Smart Grid which we need to simulate

3. **Complex packet variant** – Simplified packet variant was identified as insufficient due to its weaknesses in simulation of TCP transfers over lines with high latency and nonzero error rate. Because this scenario is often used in Smart Grids when DTS are sending data over unreliable GPRS lines, we had to design a new approach that is for Smart Grid simulation more efficient than existing network simulators and supports required communication scenarios. Results of our work in terms of architecture design and network layers abstraction are described in previous sections.

Performance differences between mentioned variants are compared in a Table 2 in the Appendix Section.

4.2.4 Design of the communication interface for the network simulator

Our goal is to enable integration of various simulation modules written in different programming languages, therefore we had to design a communication interface that connects the central timeline, which controls the simulation run, with a simulation module. Our network simulator is implemented in .NET and in this section we introduce more of its architecture.

Our simulator is represented by a *NetworkSimulatorCore* class that implements *INetworkSimulatorCore* interface. The only required method is *ProcessEvent* method, that accepts an instance of a *SimulationEvent* class and returns a list of *SimulationEvent* instances to be stored in the central timeline.

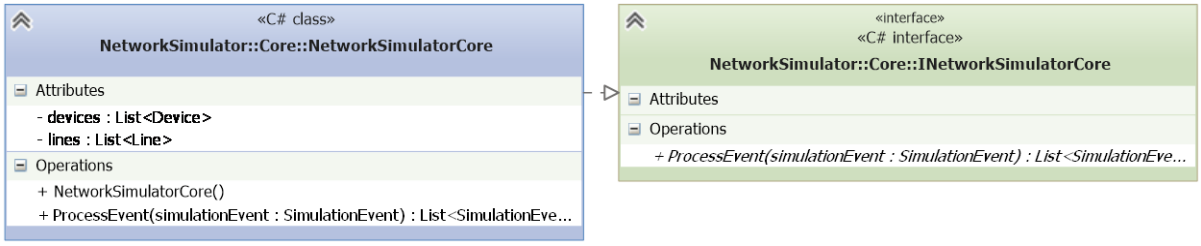


Figure 2: Class diagram of the simulator core

This simulator also comes with a variety of simulation events illustrated in Figure 3.

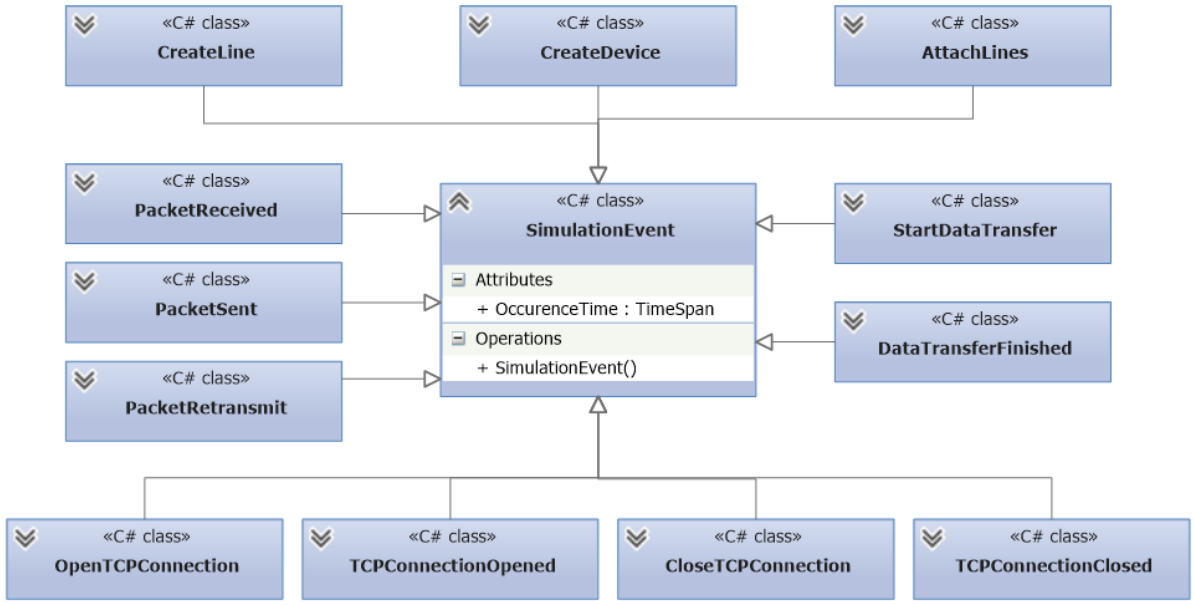


Figure 3: Class diagram of simulation events

To implement a platform independent communication interface we wrapped the *NetworkSimulatorCore* class into a command line application which hosts lightweight HTTP server using Open Web Interface technology. This allows us to make our simulation module available via platform independent HTTP REST web service.

Table 1 shows example of a request and response message via REST communication interface.

4.2.5 Validation strategy using INET simulator

As part of the network simulator development we had to create a validation environment to compare simulation results of the developed simulator with an existing network simulator solution to determine the accuracy and performance of our simulation approach in various simulation scenarios. We decided to build a second simulation environment using OMNeT++ simulation framework with its INET extension. Both the INET and our simulator were wrapped into a command line utility that accepts input parameters of a simulated line and a list of data transfers (transfer ID, data size and transfer start time). After the simulation is completed a new output file containing the list of data transfers and their finish times is generated.

To increase development velocity we implemented a test script using Power Shell, which loads all test scenarios separated into line configuration and a batch of data transfers. The script runs simulation using both simulators for every combination of a line and data transfer batch. The results are automatically collected and statistically processed using R to analyze differences between our simulator and INET.

HTTP Request sent from a timeline to the network simulation module	HTTP Response from the network simulation module

<pre>{
"$type":
"NetworkSimulator.Core.SimulationEvents.OpenTCPConnection
, NetworkSimulator.Core",
"SourceAddress": "192.168.1.100",
"DestinationAddress": "192.168.1.1",
"SourcePort": 8080,
"DestinationPort": 80,
"OccurenceTime": "00:00:02"
}</pre>
	<pre>[
	{
	"$type":
	"NetworkSimulator.Core.SimulationEvents.PacketSent,
	NetworkSimulator.Core",
	"Packet": {
	"HeaderSize": 40,
	"PayloadSize": 0,
	"Flags": 1,
	"SeqNumber": 1076383092,
	"AckNumber": 0,
	"SourceAddress": "192.168.1.100",
	"SourcePort": 8080,
	"DestinationAddress": "192.168.1.1",
	"DestinationPort": 80,
	"TransportProtocol": 0
	},
	"OccurenceTime": "00:00:02.0040000"
	},
	{
	"$type":
	"NetworkSimulator.Core.SimulationEvents.PacketReceived
	, NetworkSimulator.Core",
	"Packet": {
	"HeaderSize": 40,
	"PayloadSize": 0,
	"Flags": 1,
	"SeqNumber": 1076383092,
	"AckNumber": 0,
	"SourceAddress": "192.168.1.100",
	"SourcePort": 8080,
	"DestinationAddress": "192.168.1.1",
	"DestinationPort": 80,
	"TransportProtocol": 0
	},
	"OccurenceTime": "00:00:02.5040000"
	}
	]</pre>

Table 1: Sample request and response message illustration opening of TCP connection sent over HTTP REST communication interface

5 Lessons learned

In the initial phase of this challenging project we knew that we have to design and implement a Smart Grid simulator, but we were not experienced enough either in simulator design or network protocols. It required us to study various simulation techniques to be able to design an efficient modular simulator. This was very exciting part of the project, because we quickly implemented first prototypes of the network simulator to evaluate basic principles of a given simulation technique. To analyze the accuracy

of our simulators in the early development stage, we created a referential simulator using INET to compare simulation results from the prototypes of our simulators with a professional network simulator. We also wanted to compare performance differences, because from the beginning we were trying to answer the essential question - whether we should implement our own network simulator or integrate an existing one into our Smart Grid simulator.

We evaluated the costs related to the integration of the existing simulator which would be wrapped as a simulation module and loaded to our simulator together with other modules. We decided that the outcome will not be satisfactory because of problems with co-simulation and time synchronization which can be solved by a central event list shared by all modules, but in this case we identified in our previous tests that INET generates due to its simulation at the link layer high volume of events that would overload our communication interface and cause this simulation to run very slowly. Therefore we started to implement a new network simulator even though there are tens of existing solutions in a field of the network simulation.

The design of our simulator is focused on simplicity and modularity and we are aware of its limited usability in terms of the universal network simulation. This simulator is limited to the simulation of specific types of data transfers that occurs within Smart Grid communication.

With these limitations in mind we started to incrementally add a new functionality to increase simulation accuracy and after each iteration we reevaluated our approach. To implement the simulation of the TCP protocol we needed to become familiar with a detailed specification of this protocol which was a time intensive task due to a feature set of this protocol.

The current state of this project is that our industrial partner Mycroft Mind implemented a shared timeline component according to our common design and uses it to run simulation with their own set of simulation modules focused on application protocols for a Smart Grid. In our network simulator we add new features of the TCP protocol and evaluate their impact on a certain data transfer scenarios. We have achieved a basic level of accuracy in our simulation, but we still need to optimize the congestion control mechanisms and retransmission timers.

In this project we learned that in a simulation of a complex system every detail in the behavior of a component has to be precisely evaluated before being abstracted, approximated or omitted.

6 Conclusion

Before entering the next phase of a Smart Grid deployment following the modelling phase, the Smart Grid model needs to be validated with appropriate techniques, such as simulation. In this paper we evaluate various simulation techniques and identify the discrete event simulation as the appropriate one for Smart Grid simulation. The contribution of our work is the architecture design of a modular Smart Grid simulator, together with the design and implementation of a network simulation module optimized for the simulation of Smart Grid communication scenarios. This paper realizes this contribution and discusses the challenges encountered during the design, implementation and validation of the simulator and its communication module.

7 Acknowledgements

We would like to thank our industrial partners, especially the Mycroft Mind company and its head Filip Prochazka, for the fruitful discussions and valuable inputs that made this paper possible. We also would like to acknowledge our colleagues from the Laboratory of Software Architectures and Information Systems, especially Svatopluk Novák for his implementation of the reference INET simulator and Andrea Vašeková for the design and implementation of an early simulator prototype and valuable discussions on the simulator design.

8 References

[1] GEŠVINDR, David - BÜHNOVÁ, Barbora - ROSECKÝ, Jan. Overview of Research Challenges towards Smart Grid Quality by Design. In Proceedings of the 2014 Federated Conference on Computer Science and Information Systems. Neuveden : IEEE, 2014. ISBN 978-83-60810-58-3, 1497–1504-8 s. 9.9.2014, Warsaw, Poland.

[2] OMNeT++ Community: What is OMNeT++?. Available online at http://omnetpp.org/home/what-is-omnet.

[3] OMNeT++ Community: Simulation Models. Available online at: http://omnetpp.org/models

[4] TETCOS: NetSim - Network Simulator. Available online at: http://tetcos.com/netsim_gen.html

A Appendix

Line parameterization during the test: Transfer speed 10 000 B/s, Transfers count: 10 000, Packet size: 1500 B, Short transfer size: ±500 B, Long transfer size: ±500 000 B

Transfer type	Transfer size [MB]	Transfer mix	Simulator							
			INET		Flow		Simple Packet		Complex Packet	
			Duration [s]	Events count	Duration [s]	Events count	Duration [s]	Events count	Duration [s]	Events count
No transfer overlap	4776	0% short, 100% long	1099,200	158 919 510	0,062	20 000	4,841	3 347 149	70,577	37 479 813
No transfer overlap	3590	25% short, 75% long	442,631	119 773 270	0,062	20 000	3,641	2 524 742	51,584	28 268 933
No transfer overlap	2410	50% short, 50% long	807,538	80 537 151	0,078	20 000	2,516	1 700 432	37,635	19 036 937
No transfer overlap	1256	75% short, 25% long	322,792	42 180 051	0,078	20 000	1,578	894 555	19,941	10 011 701
No transfer overlap	48	100% short, 0% long	18,457	1 977 856	0,062	20 000	0,156	50 000	1,093	552 373
Overlapping transfers	4765	0% short, 100% long	120,846	13 752 189	47,086	30 219	350,396	3 345 696	318,633	11 152 177
Overlapping transfers	3606	25% short, 75% long	100,667	11 277 356	37,342	30 969	204,975	2 536 125	235,069	8 227 417
Overlapping transfers	2413	50% short, 50% long	80,227	8 884 605	24,911	31 833	94,875	1 702 468	154,475	5 406 656
Overlapping transfers	1218	75% short, 25% long	62,376	6 152 978	14,118	33 718	25,938	867 449	73,062	2 557 171
Overlapping transfers	48	100% short, 0% long	24,406	1 090 106	1,078	33 369	1,547	50 000	10,767	376 855

Table 2: Performance comparison of different variants of the simulator implementation

Proceedings of the 2014 Federated Conference on Computer Science and Information Systems pp. 1497–1504
DOI: 10.15439/2014F318
ACSIS, Vol. 2

Overview of Research Challenges towards Smart Grid Quality by Design

David Gešvindr
Masaryk University
Faculty of Informatics
Brno, Czech Republic
Email: gesvindr@mail.muni.cz

Barbora Buhnova
Masaryk University
Faculty of Informatics
Brno, Czech Republic
Email: buhnova@mail.muni.cz

Jan Rosecky
Masaryk University
Faculty of Informatics
Brno, Czech Republic
Email: j.rosecky@mail.muni.cz

***Abstract*—Power distribution systems worldwide are entering one of the biggest transformations since their inception. The Smart Grid has become a strategic term, which is associated with promising technology enhancement on one hand, but also high investment risks on the other hand. To maximize the former and minimize the latter, power distribution companies are searching for methods and tools to simulate and compare different Smart Grid design alternatives and deployment scenarios.**

In this paper, we draw upon our involvement in the modelling and simulation of the Smart Grid in the Czech Republic, and elaborate possible directions of Smart Grid modelling and analysis research to reach the "quality by design" before the actual Smart Grid realization. The discussion then details a specific design challenge together with its context within the design of the Czech Smart Grid.

I. Introduction

THE CURRENT generation of power distribution networks faces new requirements that stimulate discussion on its future transformation. Some of the strongest triggers that the current power grid fails to keep pace with are the transition to a distributed power generation [1] and demand response optimization [2], whose both rely on mature information exchange and control. This stimulated the integration of the power distribution and information technology domains into the concept of a Smart Grid. The Smart Grid is an electricity network that can cost-efficiently integrate the behavior and actions of all users connected to it—generators, consumers and those that do both—in order to ensure economically efficient, sustainable power system with low losses and high levels of quality and security of supply [3]. The Smart Grid relies on an Advanced Metering Infrastructure (AMI) [4], [2] with a bi-directional communication channel which allows the control of electricity demand at customer level by directly or indirectly (e.g. by on-line adjusting electricity prices) controlling household appliances [2].

The deployment of Smart Grids is becoming a strategic act for many countries, forced by their legislation [5]. European Commission calls in its mandate M411 [6], released in 2009, for the standardization in the area of Smart Metering. The standards should enable interoperability of utility meters (water, gas, electricity, heat), and allow mass production of Smart Meter devices at the competitive European market. The mandate is aligned with the OPEN meter project [7], which ended in June 2011 and delivered open and public standards for smart metering, and consequent OpenNode project [8] with similar endevour regarding low-voltage monitoring.

The transition to the Smart Grid represents heavy investments (in orders of hundreds of millions of Euros) into technology that was never before deployed in such a large scale. To minimize investment risks and maximize benefits, power distribution companies are seeking new ways of Smart Grid modelling and simulation to evaluate and compare different Smart Grid architectures and deployment scenarios. This initiatives aim in each national context at a cost- and quality-effective validated solution, before mass investments into Smart Grid deployment.

Goal of the paper: In this paper, we draw upon our involvement in the modelling and simulation of the envisioned Smart Grid in the Czech Republic, and elaborate a discussion on the possible directions of Smart Grid modelling and analysis to reach the "quality by design" before the actual Smart Grid deployment.

Paper structure: The paper is organized as follows: Section II overviews the history, context and key terminology of the field; Section III discusses the specific research questions related to the modelling, simulation and analysis towards Smart Grid design; Section IV elaborates a specific research question concerned with the communication strategy discussed in the specific context of the Czech Smart Grid design; and Section V concludes the paper.

Related work: Although numerous overview papers have been published in the Smart Grid domain, the discussion of research questions and challenges towards mature Smart Grid design is not receiving systematic attention. The existing surveys instead focus on the goals, impact, benefits, risks and undelaying technology principles (architecture, communication technologies) of the Smart Grid, usually influenced by national research and deployment [4], [9], [2], [10]. The source of the information are hence the research papers elaborating individual research questions in isolation, such as for instance the papers on multi-agent approaches in a distributed Smart Grid [11], [12], [13], security-driven Smart Grid design [14], [15], [16], survivability analysis and Smart Grid self-repair [11], [17], or demand response optimization [18].

978-83-60810-58-3/$25.00

II. Background

One of the first large network utilities was built by Thomas Alva Edison in 1882 in New York. Edison promoted direct current for electric power distribution, despite the disadvantages of this approach, namely in short distance between a power plant and its customers (2.4 km was the limit of effective distance), which meant that small power plants needed to be built in customer areas. New research in Europe and America has brought electric motors and transformers working with alternating current and with the help of Edisons major opponent George Westinghouse, the previously used direct current was replaced with the alternating current, which persisted till today. Alternating current could be efficiently transmitted over long distance and hence allowed power plants to be built outside inhabited areas, closer to input resources.

The process of power transmission nowadays involves step-up transformers that transform electricity into very high voltage to decrease transmission losses. Electricity is then transmitted over hundreds of kilometers of transmission network. Before distribution to end customers, the voltage is decreased by a series of step-down transformers, which act as gates between high voltage *power transmission* and low voltage *power distribution networks*.

In the Czech Republic the *power transmission network* consists of power plants, first-level substations and high-voltage power lines. The first-level substations and power plants are connected using 400 kV and 220 kV lines. There is high flow of energy and therefore very high voltage is used to lower the electric current, which leads to lower losses during transmission. The first-level substations produce also 110 kV intended for power distributors.

The Czech *power distribution network* consists of 110 kV lines, second-level substations transforming 110 kV to 22 kV or 35 kV for connected third-level substations. Multiple third-level substations are situated in towns and produce well-known 230V/400V used by end customers.

The current generation of power transmission and distribution network has been designed for centralized electricity generation in a small number of large power plants. Nowadays, this concept is being replaced with a more distributed electricity generation thanks to an increasing amount of predominantly renewable energy sources. European Union supports renewable energy sources in its Energy and Climate Change Policy [20] with the goal to achieve 20% renewables share in the European energy mix until 2020.

The new approach, however, brings up serious issues: (1) Since the grid load changes along the day and since there is no efficient way to centrally store the surplus of generated electricity for later use or to rapidly increase the demand of electricity, power plants have to adjust their electricity production to meet the expected demand. Immediate production change is very complicated for most types of power plants, so both the consumption and the production of the renewables have to be predicted in advance, which brings additional risks to the whole process of electricity production and distribution. Moreover, thoughtless connection of many uncontrolled renewables (e.g. photovoltaic panels) may (2) damage the current grid or (3) cause significant technical losses, because the amount of generated electricity may in specific areas exceed its demand and overload the related part of the grid.

The first step towards effective management of power distribution is Smart Metering, which measures and collects power consumptions of individual customers. In particular, the deployment of Smart Metering can provide the following inputs for smart power distribution management:

a) Collection of power consumption data: Without deployment of Smart Meter units, the power consumption measurements are automatically collected only during electricity transmission in large substations. In the Czech Republic, moreover, only very large customers have similar devices already deployed. Collection of more data from individual consumers and all substations is a necessary condition of reliable predictions, which can help to locally balance the grid load.

b) Early fault detection: Current generation of power grids can only detect failures in substations that are connected via computer network to a control center that monitors the grid. It is also possible to detect broken high-voltage wires. But when it comes to individual customers, it does not provide means to detect power outages at individual customer level, because such an outage cannot be detected directly from the substation. In the Czech Republic, this type of outage is usually reported by the customer via a phone call at distributors service line. Nowadays, to detect this type of outage automatically, each monitored customer needs to be equipped with a device that periodically sends hearth beat signals to the substation. Smart meter measurements sent to a substation can substitute the hearth beat signaling.

The current state of the practice in power transmission and distribution implies numerous challenges for the Smart Grid design and deployment, many of which could be adressed with multi-agent approaches. The next sesion discusses these challenges and related research questions in detail.

III. Smart Grid modelling, simulation and analysis

To minimize the risks associated with the Smart Grid deployment and maximize its quality across multiple quality attributes, possible modelling and simulation techniques to evaluate and compare different Smart Grid design alternatives before the actual deployment are becoming of high interest to power distribution companies. This section summarizes the specific research questions towards Smart Grid design that we identified with our industrial partners, and aims at opening the discussion on the newly establishing research field of Smart Grid design support, to which the computer science, and multi-agent research in particular, can strongly contribute. The areas include:

Fig. 1. Map of the Czech transmission network, where red labels are the major power plants, red lines are 400 kV lines, purple dots are large substations and blue lines are 220 kV lines [19].

A. Prediction of the expected consumption
B. Load control
C. Energy backup and storage
D. Hardware infrastructure design
E. Smart grid topology design
F. Decision on the types of information to be exchanged and transferred
G. Evaluation of Smart Grid communication strategies
H. Quality assessment of different Smart Grid design options

A. *Prediction of the expected consumption*

Smart metering can significantly contribute to the creation of a new global power consumption model, recording power consumption even at an individual customer level. Such a model can be used for the prediction of expected electricity consumption. As mentioned in [2] a valid consumption-prediction model is much more complex than it used to be before, because of customer life style change. Also, a need to predict locally-dependent consumption and ability to react on lower electricity prices arises together with the requirement on local load-balancing.

We identified the following research questions associated with this topic:

- How to predict power consumption in both normal and crisis situations?
- What is the appropriate level of detail on which the consumption should be studied? How to cluster the Smart Grid architecture into consumption-relevant levels for analysis?
- What trends and technologies may significantly affect the validity of current prediction models? How to predict the consumption associated with the increasing use of electric automobiles?

B. *Load control*

The goal of the load control is to lower the gap between the base load and the daily peak load in each locality. This can be achieved by shifting power usage to off-peak hours, in particular to balance production of local renewables. There are multiple advantages of this approach: (1) It decreases the maximum power generation and transition capacity required to avoid blackouts; (2) It avoids extensive stopping and starting of power generating units by shaping the power usage to remain relatively constant over time [2]; (3) It helps avoiding electricity overflow from low voltage to high voltage and related problems mentioned above. This can be achieved either manually by motivating the customers to shift their power usage via variable pricing models or automatically by remote power distributor signaling that controls customer equipment.

In the Czech Republic, a combination of both mechanisms is used for households with water boilers which are controlled remotely via HDO signals[1]. The signals are broadcasted together with a code of the controlled area, so that only HDO switches in the targeted area are affected. Nevertheless, since these groups are fixed, one cannot smoothly control the load-balance, hence this solution works only partially and cannot be considered sufficient for the uprising problems with local renewable energy sources.

[1]HDO is a shortcut that stands for “Massive Remote Control” in Czech.

There are algorithms that can optimize the power demand at the customer level based on the electricity price. One of them is described and validated in [18]. However, to validate such a demand response model in a global context, we need to step-up in the hierarchy and model many interconnected customers in the grid and mainly the behavior of the distributor, who sends impulses (in form of real-time price or direct control of appliances) to increase or decrease the demand.

The research questions that could be answered with the help of a complex demand response model are:

- Are the current communication channels suitable for reliable control of customer appliances or real-time delivery of the price? How does the system behave in case of a broken communication line?
- How effective is the proposed demand response model in terms of lowering the gap between the base load and the peak load? What is its response time?
- What is the overall reliability and safety of the proposed solution? What happens if the load control mechanism misbehaves? E.g. intelligent agents start to spread invalid information to other agents in the grid which leads towards a snowball effect? What protection needs to be implemented to ensure grid survivability in such a case?

C. Energy backup and storage

One of the consequences of high difference between the base and peak load in current power grids can be the need for large-scale (to balance the whole grid) and small-scale (to balance individual areas) energy storage. As for the large-scale energy storage, according to Electric Power Research Institute the pumped-storage hydroelectricity provides more than 99% of bulk storage capacity worldwide [21] with the efficiency that varies between 70% and 80%.

Another approach to energy storage is the agent based micro-storage introduced by Vytelingum et al. [22]. This approach models the use of small capacity batteries (4 kWh) deployed in households and accompanied with an intelligent agent, which may be an integral part of a Smart Meter. The agent buys electric energy when it is cheap and stores this energy in batteries for its use during peak time when energy price increases. According to the study, this system can achieve savings of up to 13% for an average customer using a 4 kWh battery. Moreover, in case of power outage, the system can work as an energy backup. Other approaches involve electromobile batteries or local hydrogen production.

The research questions associated with this domain are:

- How to predict required energy storage capacity in case of successful Smart Grid deployment?
- Shall households employ local small-capacity energy storage? Will their deployment affect the overall demand-response strategy? Should they be strictly controlled by the utility provider or should they autonomously react on incentives (lower energy prices)?
- How will the demand response problem be affected by growing share of electro vehicles which are charged mostly over night or during working hours? Should be the chargers controlled remotely? How to balance the need of the user to charge the vehicle and the need of the grid to govern the demand? How to predict electric vehicle usage and its recharge power consumption?
- How to solve the problem of energy surplus when the demand response mechanism fails?

D. Hardware infrastructure design

The goal at least in the Czech Republic is to minimize hardware infrastructure investments and construction costs, and hence reuse as many existing infrastructure elements as possible when transiting to the Smart Grid. The suitability of every reused component should be individually assessed together with its parameters such as component dependencies, reliability, replacement costs and other associated limitations comparing to modern alternatives. Such an evaluation process, which is in fact multi-objective optimization, may be very slow and complex when done manually. Therefore, it is beneficial to employ a Smart Grid model with all relevant components modelled together with their dependencies and parameters. The model can be then used for semi-automatic or automatic evaluation of different versions of hardware components and their associated risks and benefits. The automation of this process may lead to more complex analysis of the model in a larger scale, which may identify previously hidden solutions that can be easily validated with running simulation on top of the grid models.

This approach is suitable for the modelling and validation of Smart Meter features, communication elements, and data concentrators. Even control and data-processing software can be evaluated in this way.

The most relevant research questions are:

- What design strategies can be used to maximize the quality of the Smart Grid under the constraint of legacy infrastructure?
- What hardware infrastructure changes will imply the highest ratio of quality increase to implementation costs (i.e. quality/cost)?

E. Smart grid topology design

The overall Smart Grid quality and behavior depends not only on the hardware components, but also on their organization into the grid topology. The topology, which determines the logical architecture of the grid, can take advantage of different levels of centralization (e.g. customer data from a given region managed through a single substation), hierarchical structure (e.g. substations governed by parent substations) and redundancy (to minimize the critical chain effects and maximize grid survivability in case of local outage).

The relevant research questions towards the logical architecture of a Smart Grid are the following:

- What shall be the level of centralization within the Smart Grid?

- Shall the substations and other grid components be organized in a hierarchical manner? If yes, how many levels should be used?
- What level of homogeneity/heterogeneity (with respect to the smart logic within the grid nodes) is most beneficial?
- What are the critical chains of the Smart Grid with respect to the consequences of their outage?
- What is the level of grid robustness and survivability? What are the most cost-effective topology improvements that significantly increase robustness and survivability? What is the expected benefit of self-repair mechanisms comparing to their implementation costs?

F. Decision on the types of information to be exchanged and transferred

The concept of a Smart Grid relies on the transfer of various information between grid components. The Advanced Metering Infrastructure (AMI) needs to transfer measurement results collected from Smart Meters for processing purposes, substations need to report their state to a parent substation in case of hierarchical infrastructure or broadcast their state to other substations in case of multi-agent processing.

To minimalize the traffic among Smart Grid components, the information value of different types of data needs to be understood and used to drive the identification of needless data, whose transfer should be limited. On the other hand, the critical information types shall be prioritized in the communication.

The relevant research questions include:

- How to identify and model importance of collected measurements for later processing in the Smart Grid? How this process can be automated?
- What is the ideal collection frequency of measurements from Smart Meters to satisfy the legislation needs, ensure sufficient precision of prediction models and help load balancing on one hand and lower the communication line load on the other hand?
- How to efficiently identify dependent information for producing complex measurements across more elements in the grid?
- What is the acceptable age of particular type of information for a given stakeholder?

G. Evaluation of Smart Grid communication strategies

Modelling of different communication strategies in the Smart Grid is essential for the evaluation of grid behavior in different situations. Identification of a suitable communication strategy is often hindered with a set of constraints that must be fulfilled. For instance, one of the primary restrictions in the Czech Republic is the employment of existing communication infrastructure, because significant rebuild of the infrastructure would unacceptably increase the Smart Grid deployment costs. When following the physical structure of the grid described above, we need to transfer measurements from Smart Meter units placed at customers electrical connections to a datacenter for processing, and control commands back to the elements of the grid. The problem is that a Smart Meter cannot be usually connected directly to the computer network to send all required data securely over the Internet.

Employment of the Internet connection of the customer for direct connection of the Smart Meter to the Internet would be a solution, but in terms of its availability, reliability and the legal complexity, it is merely impossible to deploy it nationwide.

There are two available groups of technologies that can be used in Smart Grids to transfer data between the elements of the grid—data transfer over power lines and wireless data transfer. Possible use and issues associated with these technologies in the Czech Smart Grid are examined in Section IV.

Relevant research questions for this area are:

- Can network modeling and network simulation tools be used in large scale as Smart Grid modeling and validation requires? What optimizations need to be done prior their use?
- What level of precision in a network simulation is required for trustworthy validation of the communication strategy?
- What communication channels and communication strategies are suitable for control channel? What are the requirements that control channels need to meet?
- Is it better to transfer raw measurements or pre-processed data? Where does the processing take place?
- How can we increase the reliability of the line? How can we get cost-effective line redundancy?

H. Quality assessment of different Smart Grid design options

The quality evaluation, prediction and optimization during Smart Grid design are critical cross-cutting concerns to all the research issues discussed above. The Smart Grid quality is typically understood in terms of the grid reliability, security and efficiency, and considered along with its environmental and energy sustainability and cost.

The reliability engineering in the Smart Grid domain aims at incorporating autonomous control actions to enhance reliability by increasing resiliency against component failures and natural disasters, and by minimizing frequency and magnitude of power outages subject to regulatory policies, operating requirements, equipment limitations, and customer preferences [23]. This includes the techniques for automated failure detection, isolation and restoration (FDIR), and terms such as grid survivability and robustness.

The security requirements for a Smart Grid are mainly the authenticity, integrity, and privacy of metering data, along with the safety and availability of the grid [24]. The Smart Grid design shall aim at resiliency against malicious attacks through better physical security and cybersecurity, to protect the infrastructure, energy asset and data of the grid.

The efficiency of a Smart Grid refers to the infrastructure utilization on both the energy-transfer and data-transfer level. The new research questions emerge mainly in the context of data-transfer efficiency, which is specifically concerned with

the level of utilization of communication channels and latency of data delivery.

The research questions concerned with the quality assessment and optimization include:

- Which of two or more Smart Grid design options is better with respect to a specific quality attribute?
- How shall a specific Smart Grid design alternative be changed to optimize a specific quality attribute (under given constraints, such as the cost)? E.g. to optimize the time needed for failure detection, isolation and grid restoration.
- What is the quality trade-off of different quality-related strategies? E.g. what is the security improvement and efficiency degradation of a specific security strategy?
- What is the context (characteristics of the settings) in which a specific quality-improving strategy yields highest benefits?
- Can the Smart Grid reach a specific unsafe condition (i.e. a state violating safety specification)?
- What are the critical components of the Smart Grid with respect to a specific quality attribute? E.g. what are the most critical components with respect to grid security or survivability? What communication links have the highest impact on data-transfer latency?
- What is the level of satisfaction of individual stakeholders, including the energy distributors, customers and public organizations?

Following on from these research questions the next section describes in detail various communication strategies that are being currently evaluated by Czech power distribution companies for potential deployment in their Smart Grid projects.

IV. Situational study of the communication strategy for the Czech Smart Grid

Together with our industrial partner, the Mycroft Mind company, which is a leading expert in the field of Smart Grid simulation in the Czech Republic, we are researching new ways of Smart Grid modelling, simulation and analysis that can be integrated into their analytical tool called Grid Mind.

One of the biggest challenges in Smart Grid deployment in the Czech Republic is the communication strategy. The Advanced Metering Infrastructure (AMI) produces waste amount of data that needs to be transferred to a datacenter of the power distributor for processing. Every day the grid of the CEZ power distributor is expected to produce more than 340 Million measurements. The key concern is that the current communication channels will not be able to transfer the measurements in time. Another challenge is to process the measured data at real time, which is necessary for effective demand response optimization. Last, there is a concern that the control channels will not be able to deliver the control commands with the necessary level of reliability.

Our colleagues at the Faculty of Informatics, Masaryk University, Brno, were involved in the research examining what transfer characteristics can be expected with different versions of GPRS. We now look at this problem in a broader way. Together with Mycroft Mind we are building a modelling and simulation environment that is planned to be used for modelling, simulation and validation of various communication strategies in the Smart Grid, whose key characteristics are elaborated in the rest of this section.

There are two available groups of technologies that can be used in a Smart Grid to transfer data between the elements of the grid—data transfer over *power lines* and *wireless* data transfer.

The Power Line Carrier (PLC) technology stands for a family of communication protocols and technologies that transfer data over *power lines*. Different PLC technologies vary in frequency ranges used as transport medium for data: NPL (Narrow Power Line) uses frequencies between 3 KHz and 148.5 KHz, BPL (Broadband Power Line) uses higher frequency range between 2 MHz and 50 MHz. The quality of the data transfer is significantly dependent on the properties of the power line, such as the path attenuation and noise in case of medium voltage lines connecting primary to secondary substations. The number of parameters that need to be evaluated increases when low voltage lines that connect secondary substations and Smart Meter devices are used. More detailed characteristics of PLC technologies can be found in [25].

In the low voltage network there are two types of devices: central low voltage unit (CLU) and remote low voltage unit (RLU). CLU is located at low voltage substations usually with a few hundreds of connected customers. RLU is built into a Smart Meter and may act also as a repeater in order to extend the action radius of the PLC signal. The PLC communication medium is composed of sub networks defined by CLU. To avoid collisions, access to PLC medium has to be controlled, therefore CLU controls all the traffic and RLU are allowed to communicate only in case they respond to CLU's request.

PLC was chosen as a suitable technology for centralized collection of measurements from Smart Meters. The benefit is that it uses existing power lines which decreases deployment costs and provides sufficient communication channel for collection of power consumption measurements. The deployment of this technology is currently being validated using almost 40,000 Smart Meter units deployed in four Czech towns [26]. The problem is how to transfer the collected data from substations.

The latter group of technologies usable for data transfers in the Smart Grid is *wireless* transfer. There are many available wireless communication technologies that can be effectively used for measurements transfer to a data center for processing. We discuss some of the technologies usable over large distance and explain their advantages and disadvantages below.

a) 2G GPRS/EDGE: In the Czech Republic, the coverage of public cellular networks is approaching 99% of area. 2G cellular networks provide data services in the speed of kbps or tens kbps. These data services are optimally tailored for automated meter readings. Mobile operators offer data-only SIM cards for industrial use with lower communication priority. No investments into network infrastructure are needed

for cellular network-based AMI solutions. Transferring data over 2G cellular network from each individual Smart Meter unit would not be cost effective in terms of running costs paid to mobile operator, therefore we propose to use this technology to transfer collected data from substation. It is necessary to model such a situation and validate if the transfer speed of 2G networks is sufficient.

b) 3G UMTS: 3G cellular networks are extending the functionality of traditional 2G networks, allowing data transfers at significantly higher speeds. With HSPA extension peak data rates up to 14 Mbit/s in the downlink and 5.75 Mbit/s in the uplink are supported. Data rates of 3G networks are fully sufficient for AMI purposes. In the Czech Republic, most of the 3G coverage is in densely populated areas. According to the data provided by the Czech Telecommunication Office, the mobile operator with the best 3G signal coverage covered at the end of year 2012 only 48% of the Czech Republic [27]. Substations in uncovered areas will have to rely on slower 2G networks.

c) 4G LTE: LTE cellular network rapidly increases transfer speeds up to 100 Mbit/s. Mobile operators started to deploy this communication technology in the Czech Republic at the end of 2013. The Vodafone mobile operator selected this technology as a complement to their 3G network which will not be extended anymore. They would like to primarily upgrade 2G EDGE connection in places uncovered by HSPA to cover 93% of the Czech population with 3G or 4G connectivity until December 2014. Such a rapid deployment of LTE as a replacement of 2G network should be considered in evaluation of Smart Grid communication model in case of 2G networks insufficient speed.

d) WLAN/WiFi: Broadband wireless technologies based on the IEEE 802.11 standard have found broad acceptance worldwide for wireless area networking. The use of WLANs for AMR has been frequently reported. In the US, several cities (e.g. Santa Clara, CA) already use WLANs to automatically collect readings from electricity, gas, and water meters [28]. Using 802.11s technology supporting mesh may provide a technically attractive solution for AMR [28]. Use of public WLAN in municipalities, if available, may solve problems with connection availability at substations, but considerable number of substations is out of reach of WLANs. Using WLAN as primary connection technology for measurements collection is less reliable than PLC.

e) WiMAX: Worldwide Inter-operability for Microwave Access is a wireless broadband access technology which has been designed to provide transfer speeds up to 72 Mbit/s in both directions. Communication range varies between few kilometers in case of non-line of sight conditions and 50 km in case of line of sight conditions. Deployment of a dedicated WiMAX network solely for the purpose of an AMI likely cannot be justified. In those areas where public WiMAX services are available (e.g. municipal networks), WiMAX may principally be an option [28].

f) ZigBee: ZigBee standards-based protocols provide infrastructure for wireless sensor network applications. ZigBee is capable of creating a mesh network, which provides additional capabilities—increased redundancy, self-configuration and self-healing. Situation of building nationwide dedicated network using ZigBee is the same like in case of WiMAX. Possible use of ZigBee wireless network in terms of Smart Metering is connection of individual sensors where the power line does not meet quality requirements of PLC based protocol. Another use may be peer-to-peer connection between substations to increase redundancy or capacity of Internet connection (e.g. EDGE).

g) Satellite systems: Satellite system are suitable for sending measurements in areas where no suitable terrestrial infrastructure is available. Today, there is a number of satellite operators offering data transfer services. For instance Iridium service, which uses low earth orbit satellites or SpaceChecker that uses geostationary satellites. Deployment in areas covered by other communication infrastructures needs to be evaluated, because costs are expected to be significantly higher than use of other terrestrial communication technologies.

The infrastructure that is currently examined in the Czech Republic combines PLC technology to transfer measurements from Smart Meters to a substation. Each substation is collecting data usually from a few hundreds Smart Meters. The data concentrator unit, which is deployed at substations, can process data locally which opens new possibilities in building more decentralized solutions. Each data concentrator is equipped with a cellular network modem, in the Czech Republic usually EDGE technology, and sends data to a data center. This approach brings significant improvement in terms of connection costs reduction but increases the demands on speed and reliability of Internet connection available at substations, because more collected data needs to be transferred from a single point.

We were provided with the operational measurements of data transfer speed and latency of currently used GPRS communication channel in the test deployment of Smart Grid in the Czech Republic. The used low-priority industrial SIM cards can utilize only 10% of the communication capacity of a mobile base transceiver station (BTS), therefore under heavy load they operate only at the speed of few Kbits per second. Provided measurements show problems with connection handling across different levels of the network stack. If there is a network connection that is not used for a longer period of time, this connection is disconnected by the BTS at the physical level, but the TCP connection remains open. Any later attempt to transfer data via this broken TCP connection results in a communication failure and a need to reopen the connection which takes a few seconds to establish. The discussed problems illustrate the need to build a valid simulation model of the network communication in the Smart Grid. To be able to handle the high complexity of this model, we propose together with our industrial partner Mycroft Mind decomposition of this model into three interconnected simulation models. The

first model describes the simplified behavior of the physical layer, the second model describes the simplified version of the transport protocol and the third model describes used application protocols. The Grid Mind simulation environment utilizes the discrete event simulation and proposed models to evaluate different communication channels, protocols and communication strategies.

Constraint in a form of available Internet connection at the locality of the substation is one of the key problems that needs to be evaluated using correct modelling and simulation approaches.

V. Conclusions

To minimize the risks associated with the Smart Grid deployment and maximize its quality across multiple quality attributes, possible modelling and simulation techniques to evaluate and compare different Smart Grid design alternatives before the actual deployment are becoming of high interest. In this paper, we elaborated a discussion on the possible directions of Smart Grid modelling and analysis to reach the "quality by design", and presented specific research questions towards Smart Grid design that we identified with our industrial partners. One of the research questions, concerned with the communication strategy, was then discussed in detail and linked to the specific situation in the Czech Republic. The communication strategy evaluation is at the same time our ongoing research aim, which will be in the future extended in the direction of the research questions discussed in this paper.

Acknowledgement

We would like to thank our industrial partners, especially the Mycroft Mind company and its head Filip Prochazka, for the fruitful discussions and valuable inputs that made this paper possible.

References

[1] J. Bremer and M. Sonnenschein, "A distributed greedy algorithm for constraint-based scheduling of energy resources," in *Proc. of FedCSIS'12*. IEEE, 2012. ISBN 978-1-4673-0708-6 pp. 109–115.

[2] Y. Simmhan, S. Aman, B. Cao, M. Giakkoupis, A. Kumbhare, Q. Zhou, D. Paul, C. Fern, A. Sharma, and V. Prasanna, "An informatics approach to demand response optimization in smart grids," Computer Science Department, University of Southern California, Tech. Rep., 2011.

[3] EU, "Standardization mandate to european standardisation organisations (ESOs) to support european smart grid deployment," March 2011, last retrieved 2014-04-21. [Online]. Available: http://ec.europa.eu/energy/gas_electricity/smartgrids/doc/2011_03_01_mandate_m490_en.pdf

[4] X. Fang, S. Misra, G. Xue, and D. Yang, "Smart grid – the new and improved power grid: A survey," *Communications Surveys Tutorials, IEEE*, vol. 14, no. 4, pp. 944–980, 2012. doi: 10.1109/SURV.2011.101911.00087

[5] EU, "Single market for gas & electricity: Smart grids," July 2013, last retrieved 2014-04-21. [Online]. Available: http://ec.europa.eu/energy/gas_electricity/smartgrids/smartgrids_en.htm

[6] EU, "Standardisation mandate to CEN, CENELEC and ETSI in the field of measuring instruments for the development of an open architecture for utility meters involving communication protocols enabling interoperability," March 2009, last retrieved 2014-04-21. [Online]. Available: http://ec.europa.eu/energy/gas_electricity/smartgrids/doc/2009_03_12_mandate_m441_en.pdf

[7] OPEN meter, July 2012, last retrieved 2014-04-21. [Online]. Available: http://www.openmeter.com/

[8] M. Alberto, R. Soriano, J. Gotz, R. Mosshammer, N. Espejo, F. Lemenager, and R. Bachiller, "OpenNode: A smart secondary substation node and its integration in a distribution grid of the future," in *Proc. of FedCSIS'12*. IEEE, 2012. ISBN 978-1-4673-0708-6 pp. 1277–1284.

[9] R. Hassan and G. Radman, "Survey on smart grid," in *Proc. of IEEE SoutheastCon'10*, 2010. doi: 10.1109/SECON.2010.5453886 pp. 210–213.

[10] W. Wang, Y. Xu, and M. Khanna, "A survey on the communication architectures in smart grid," *Computer Networks*, vol. 55, no. 15, pp. 3604–3629, 2011. doi: 10.1016/j.comnet.2011.07.010

[11] S. Amin and B. Wollenberg, "Toward a smart grid: power delivery for the 21st century," *Power and Energy Magazine, IEEE*, vol. 3, no. 5, pp. 34–41, 2005. doi: 10.1109/MPAE.2005.1507024

[12] M. Pipattanasomporn, H. Feroze, and S. Rahman, "Multi-agent systems in a distributed smart grid: Design and implementation," in *Proc. of PSCE'09*, 2009. doi: 10.1109/PSCE.2009.4840087 pp. 1–8.

[13] S. D. Ramchurn, P. Vytelingum, A. Rogers, and N. Jennings, "Agent-based control for decentralised demand side management in the smart grid," in *Proc. of AAMAS'11*. ACM, 2011, pp. 5–12.

[14] P. McDaniel and S. McLaughlin, "Security and privacy challenges in the smart grid," *Security Privacy, IEEE*, vol. 7, no. 3, pp. 75–77, 2009. doi: 10.1109/MSP.2009.76

[15] H. Khurana, M. Hadley, N. Lu, and D. Frincke, "Smart-grid security issues," *Security Privacy, IEEE*, vol. 8, no. 1, pp. 81–85, 2010. doi: 10.1109/MSP.2010.49

[16] A. Metke and R. Ekl, "Security technology for smart grid networks," *IEEE Transactions on Smart Grid*, vol. 1, no. 1, pp. 99–107, 2010. doi: 10.1109/TSG.2010.2046347

[17] D. S. Menasché, R. M. Meri Leão, E. de Souza e Silva, A. Avritzer, S. Suresh, K. Trivedi, R. A. Marie, L. Happe, and A. Koziolek, "Survivability analysis of power distribution in smart grids with active and reactive power modeling," *SIGMETRICS Perform. Eval. Rev.*, vol. 40, no. 3, pp. 53–57, Jan. 2012. doi: 10.1145/2425248.2425260

[18] A. Conejo, J. Morales, and L. Baringo, "Real-time demand response model," *Smart Grid, IEEE Transactions on*, vol. 1, no. 3, pp. 236–242, 2010. doi: 10.1109/TSG.2010.2078843

[19] CEPS, "Schéma rozvodné sítě v ČR (schema of Czech transmission network)," 2013, last retrieved 2014-04-21. [Online]. Available: http://www.ceps.cz/CZE/Cinnosti/Technicka-infrastruktura/PublishingImages/Mapa_siti_CZ.PNG

[20] EU, "Communication from the commision to the european council and the european parlament - an energy policy for europe," January 2007, last retrieved 2014-04-21. [Online]. Available: https://www.europia.eu/Content/Default.asp?PageID=412&DocID=13922

[21] EPRI, "Electricity energy storage technology options," December 2010, last retrieved 2014-04-21. [Online]. Available: http://www.epri.com/abstracts/Pages/ProductAbstract.aspx?ProductId=000000000001020676

[22] P. Vytelingum, T. D. Voice, S. D. Ramchurn, A. Rogers, and N. R. Jennings, "Agent-based micro-storage management for the smart grid," in *Proc. of AAMAS '10*, 2010, pp. 39–46.

[23] K. Moslehi and R. Kumar, "A reliability perspective of the smart grid," *IEEE Trans. on Smart Grid*, vol. 1, no. 1, pp. 57–64, June 2010. doi: 10.1109/TSG.2010.2046346

[24] D. von Oheimb, "It security architecture approaches for smart metering and smart grid," in *Proc. of SmartGridSec'12*, ser. LNCS, vol. 7823. Springer, 2013. doi: 10.1007/978-3-642-38030-3_1 pp. 1–25.

[25] OPEN meter, "Description of current state-of-the-art of technology and protocols - description of state-of-the-art plc-based access technology," May 2009, last retrieved 2014-04-21. [Online]. Available: http://www.openmeter.com/files/deliverables/OPEN-Meter\%20WP2\%20D2.1\%20part2\%20v2.3.pdf

[26] ČEZ, "Čez smart grids inteligentní měření," January 2014, last retrieved 2014-04-21. [Online]. Available: http://www.futuremotion.cz/smartgrids/cs/index.html

[27] CTU, "Výroční zpráva českého telekomunikačního úřadu za rok 2012 (annual report of the czech telecommunication office for 2012)," April 2013, last retrieved 2014-04-21. [Online]. Available: http://www.ctu.cz/cs/download/vyrocni_zpravy/vyrocni_zprava_ctu_2012.pdf

[28] OPEN meter, "Description of current state-of-the-art technologies and protocols - general overview of state-of-the-art technological alternatives," June 2009, last retrieved 2014-04-21. [Online]. Available: http://www.openmeter.com/files/deliverables/OPEN-Meter\%20WP2\%20D2.1\%20part1\%20v3.0.pdf

Availability and scalability of web services hosted in Windows Azure

David Gešvindr

Masarykova univerzita - Fakulta informatiky
Laboratoř softwarových architektur a informačních systémů
Botanická 68a, 602 00 Brno
255653@mail.muni.cz

Abstract
This article aims to present the reader with principles of a software architecture design that leads toward better scalability and availability of a web service hosted in Windows Azure cloud environment. This article introduces the reader to major Windows Azure services and points out the most important differences of the cloud environment, that need to be taken in account in an early software architecture design phase so the service can effectively use advantages of the cloud especially almost unlimited horizontal scalability.

Abstrakt
Tento článek si klade za cíl seznámit čtenáře s principy návrhu softwarové architektury webové služby, jenž vedou k lepším charakteristikám škálovatelnosti a dostupnosti v cloudovém prostředí Windows Azure. Článek seznamuje čtenáře s hlavními službami Windows Azure a upozorňuje na důležité odlišnosti cloudového prostředí, které musí být brány v potaz již při návrhu architektury služby, pokud má služba efektivně využívat výhod cloudu, zejména téměř neomezené horizontální škálovatelnosti.

Keywords

Cloud computing, Windows Azure, software architecture, scalability, availability

Klíčová slova

Cloud computing, Windows Azure, softwarová architektura, škálovatelnost, dostupnost

1 Introduction

With increasing dependency of modern society on daily used web services even in emergency situations, there are more strict requirements on availability and scalability of hosted services. One of the possible solutions to increase these service characteristics is to use a cloud-based hosting with guaranteed service availability and scalability.

Together with broader use of the cloud there are emerging questions if current software architectures used for 2 or 3-tier web applications are able to use full potential of the cloud and if they are also highly available and scalable or there is need to change the architecture of the designed service to ensure this characteristics.

In first section of this paper Windows Azure cloud platform is introduced. Section 2 describes recommendations for software architecture to increase scalability of a service. Section 3 describes architecture of a service designed for efficient use of cloud environment. Section 4 evaluates this approach using load tests.

2 Overview of the Windows Azure Platform

Windows Azure is Microsoft's cloud platform providing rich set of hosting services in Microsoft's datacenters all over the world [1]. It offers hosting services that fall mostly in categories - Infrastructure as a service (IaaS) and Platform as a Service (PaaS). Developers can buy this platform's compute and storage resources to host their own web applications and services. Considering the size of datacenters of Windows Azure, there is almost unlimited amount of resources that can be provisioned. Payment is based on Pay-As-You-Go model, where the customer is billed only for currently allocated amount of compute power, storage capacity, amount of outgoing traffic from the datacenter, database size, or another billable service metric based on the pricelist of Windows Azure.

In the following sections we shortly introduce important services offered by Windows Azure and discuss their key qualitative attributes in the area of availability and scalability.

2.1 Compute Services

Windows Azure provides a range of computation services allowing developers to run their applications in the cloud environment. Biggest difference is in a level of control the developer has over the environment – Virtual Machines service provides freedom to build a custom server in contrast to the Web Sites service that offers almost a classical web hosting with minimum configuration options.

Developers can create their own virtual server using **Virtual Machine** service and vertically scale this server by choosing server configuration – number of CPU cores and RAM size. During the creation process they can choose from different available operating system images or they can upload their own hard drive file to Azure Storage and use it in the virtual machine. This service offers high degree of freedom, but also puts a responsibility for server maintenance at the customer. Typical use of this service is a hosting of a specific application server or strongly customized web server. To provide high availability according to the SLA – 99.95%, there have to be at least two instances of the virtual machine grouped together into an availability set.

Cloud Service offers lower degree of configuration freedom but also significantly lowers a required maintenance effort. In background it works almost the same way as a virtual machine. Developers can choose between 2 roles – web role and worker role, depending whether they need to host the application in Microsoft's web server Internet Information Service or run it as a stand-alone process. Application they build is during publication packaged into an installation package together with its configuration and is uploaded to Windows Azure. Windows Azure infrastructure then allocates required computation resources (Fabric Controller creates virtual server from unified server image depending on a selected role, deploys published application to a freshly created server and configures network infrastructure to distribute load between instances of the service if there is more than one). Newly created instance of virtual server can be customized by a startup script, which is a part of the installation package and can reconfigure OS and install another applications. The main disadvantage of cloud service is absence of persistent storage, because the cloud infrastructure never migrates existing virtual server, it always creates new instance based on the application package. Therefore all application data should be stored in Azure Storage or relational database. Cloud Service can scale vertically – increase computing power of a single node or horizontally by increasing a number of nodes and distributing load between them.

For web services that do not require any specific configuration changes in a web server it may be cost efficient to use the **Azure Web Sites** service. This service offers support for

ASP.NET, PHP, Node.JS and Python. Web sites can be deployed directly from source controls with support for Git, Bitbucket, CodePlex and Team Foundation Service or via FTP. Their content is stored in the Windows Azure Storage and is loaded directly by web servers, which can run in shared or reserved mode. In a shared mode websites are running on a virtual machine that is concurrently hosting many other websites of different customers in a slightly modified version of Internet Information Service. User in this mode pays for every hosted site. To guarantee better performance it is possible to create instance that is reserved only for one customer and can host up to 500 web sites. Reserved mode is billed in the same way as a virtual machine independently on the number of hosted web sites. Web sites in a reserved mode can be scaled vertically by selecting small, medium or large virtual machine instance and also horizontally by increasing the number of web servers serving the same content.

2.2 Scalability and availability of compute services

Benefit of the cloud is that a compute power can be easily increased when it is needed and decreased when the application is idle or under lower load. There are two approaches how a service can be scaled – vertically and horizontally. [2]

The vertical scaling increases available resources of a single computation node – it adds more CPU cores and RAM. This is done by changing a configuration of the virtual server. Advantage of this approach is that almost every application that can utilize mode CPU cores can benefit from this type of scalability without any needs to change the application. Disadvantage is that we have to stop the virtual server to do this configuration change and this type of scalability does not increase an availability of the hosted application. In Windows Azure we are limited because the larges instance has only 8 CPU cores. To scale the application behind this limit we have to use the horizontal scalability.

When we scale application horizontally we add more compute nodes and distribute the load between them. This approach works correctly with stateless applications, which are mostly web applications so a load balancer can distribute user requests independently on an inner state of the application on the specific node. Not every application can benefit from the horizontal scalability. Applications that work with the inner state have to be updated to store their inner state in a storage shared between all nodes using Azure Storage, SQL Database or Caching Service. Advantage of the horizontal scalability is that we can add more nodes or remove some of them without an interruption of the service running on existing nodes, simply by reconfiguring the load balancer. We can effectively use the horizontal scalability to adjust the reserved compute power to meet current load requirements.

Availability of compute services is at least 99.95% according to SLA with minimum of 2 deployed instances. When one of the instances fails, load balancer detects the missing instance and stops sending requests to the missing virtual server. When using the Virtual Machine service, it depends on the hosted application if supports failover scenarios.

2.3 Storage Services

Windows Azure offers specialized services for storing and retrieving data. Windows Azure storage works as a storage for binary data, tabular data (No-SQL approach) and queue messages. Current web services very often use SQL databases, which can be hosted in a second storage service named SQL Database.

Windows Azure Storage is a key service for storing data in Windows Azure cloud providing almost unlimited storage capacity. Depending on a type of stored data user can create different storage containers – Blob Storage for storing large amounts of unstructured text or binary data such as images or video.

Table Storage offers No-SQL functionality for applications that need to work with large volumes of unstructured data. Data in the table storage are stored as rows in a large table without a defined schema as required in SQL databases. Each row has two important fields that together uniquely identify each row – Partition Key and Row Key. Each row then can contain other fields of basic data types. In such a table there can be more different types of rows. The most important attribute of this storage to consider is that data can be effectively retrieved only based on their Partition and Row Keys. Any other type of search is very inefficient and slow.

Queue Storage works as a highly scalable queue that can be used by applications to exchange small messages between different components to create indirect dependencies. Very often this queue is used as a queue for pending task to be processed by worker roles. This approach allows off-load complex calculations from a web role to a worker role, so the web role can serve higher volume of less demanding requests and slow computations are done asynchronously by worker roles, which can be then easily horizontally scaled.

Windows Azure SQL Database is a hosted version of Microsoft SQL Server database with minor modifications to provide a high availability by replication data among three synchronous replicas. This database service has advantage in full compatibility of database clients with Microsoft SQL Server. This means that an existing database can be migrated from SQL Server to the cloud with some minor changes and existing applications only has to update its connection strings to point at the hosted version of the database.

2.4 Scalability and availability of storage services

Windows Azure Storage guarantees according to SLA availability at least 99.9%. This service has a very complex multi-tier architecture [3] that helps achieve high availability and high scalability of this service. Blobs and Tables are an ISO 27001 certified managed service, which can auto scale to meet massive volume of up to 200 terabytes and throughput for all accounts created after June 7th, 2012.

An additional Content Delivery Network service can be used to increase a throughput of Windows Azure Blob Storage. The CDN works as a distributed file cache with servers spread all over the world. When a user requests a file, it is served by the closest CDN node which in case of the first request loads the file from Azure Storage and then holds it for 72 hours in its cache.

SQL Database service offers availability of hosted databases of at least 99.9%. Because this service works in a multi-tenant environment where database servers are hosting databases of different customers there are specific policies that limit the maximum number of database connections and the overall load at the database to provide all customers the same quality of the service. To scale this service up there is available a new premium version of this service, which creates a dedicated database servers without mentioned limits especially for the customer. These premium servers are fully manage by Microsoft so there is no additional maintenance effort for the customer.

3 Recommendations to increase service scalability

This section describes general principles to increase scalability of a designed service. [4]

3.1 Software components

1. **Every component of a software system should have a clearly defined purpose.** The web service should be built as a collection of interconnected software components. It is preferable to design smaller components, which are easier to

design, code and maintain. Together with increasing size of a component there is also increasing number of dependencies for a given component. It may be very difficult to efficiently identify a performance bottleneck in a very complex component.

2. **Every component should have a clearly defined communication interface.** It should be possible to simply swap an inefficient implementation of a component for a new more efficient implementation. Having a clearly defined communication interface between software components makes this process simpler and cost effective.
3. **Do not burden component with interaction mechanism responsibility.** Every software component has its responsibilities and dependencies. A mechanism for communication between components should not be implemented as a part of a component because it increases its complexity and decreases its versatility.
4. **Distribute data sources.** It is preferable to distribute data sources across different layers of the service. All data of the service can be stored centrally in a single relational database. But together with increasing number of processing nodes there is increasing load at this central database. Relational databases are in general poorly scalable so this central database becomes soon the performance bottleneck we will need to replace. We can use different storage services available in the cloud with different performance characteristics and pricing models to create an efficient and scalable multi-tier storage.
5. **Use data replication.** We can replicate data to be stored in more than one copy closer to a computation node, so it can access the data faster (e.g. cache a data object in an in-memory cache of a web server). This approach works easily for read-only data, which can be easily without any conflicts stored in many identical copies. Using the data replication for data that are continuously modified by worker roles requires implementing a synchronization mechanism that propagates changes across all nodes in the system. Depending on the target characteristics of this synchronization mechanism, it may be very difficult to implement it. In heavily used systems with many changes in replicated data there will occur many change conflicts that may be difficult to resolve in an automatic way.

3.2 Connection between software components

1. **Use explicit connectors.** Connectors that are used to connect different components together should not be an integral part of the component so they can be easily replaced by different implementation. This can be achieved by creating a technology-dependent communication interface that wraps over an inner technology independent communication interface of the component.
2. **Each connector should have a clearly defined purpose.** So far we discussed the distinct purpose of a component but this apply also for communication interfaces which should support only few required protocols and we should rather create more simpler connectors for each protocol than one complex connector.
3. **Use the simplest available connector.** There are available many different connectors for different communication protocols that support different set of functionality. Developer should choose the simplest possible connector that meets all current requirements instead of a complex connector that may be useful in future.
4. **Choose between direct and indirect dependencies.** Direct dependencies between components like a synchronous procedure call have benefits in an obvious control flow in the system. Use of indirect dependencies where requests to call procedures are sent over a shared queue to be called asynchronously has better scalability characteristics because they can be distributer among higher number of workers.
5. **Do not put application logic into connector.** We should clearly distinguish between a purpose of a component and its connector. The purpose of the connector is to

mediate communication between the component not to run an additional application logic that would increase connector's complexity and slow it down.

6. **Use scalable connectors.** To increase overall scalability of the application it is important to use a scalable communication mechanism between its key components.

4 Architecture of service designed for cloud

Based on Windows Azure service's characteristics and general recommendations for an architecture of the software service I design following high-level service architecture that is depicted in Figure 1.

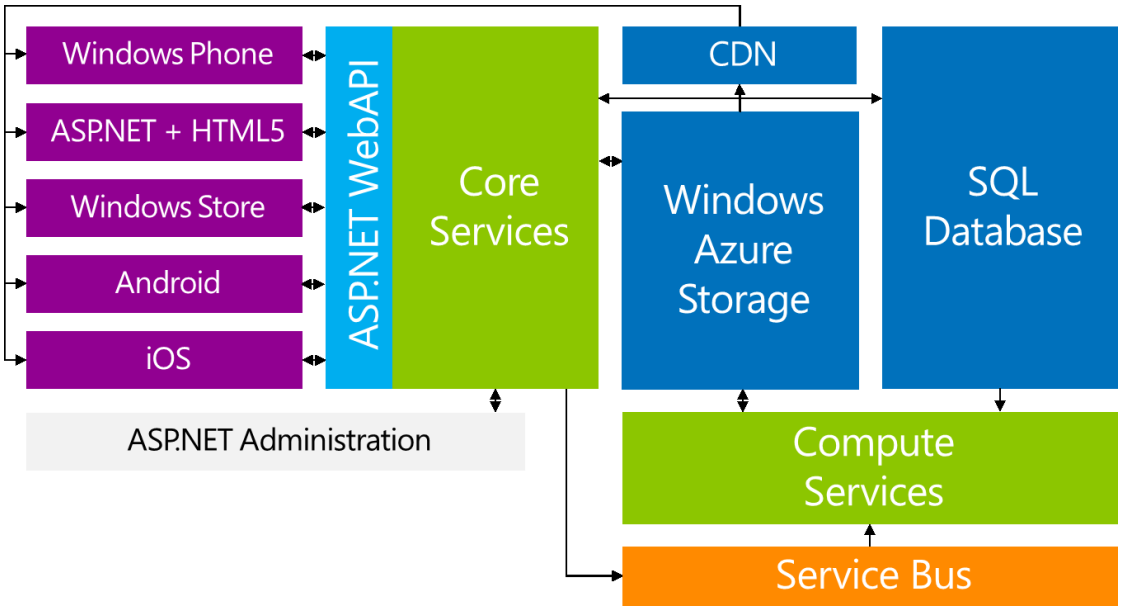


Figure 1: Proposed high-level service architecture

4.1 Storage Tier

Based on the recommendation to distribute data sources of the designed service we should use multiple services for storing data so a user load can be effectively distributed among them. Storage services should be tiered based on their scalability and availability characteristics. The service with the best scalability should process majority of user's requests. The poorly scaling service should process only a few specific requests. This means that we use the relational database service SQL Database to store most of service's structured data because it provides a mechanisms for transaction processing and can easily ensure consistency of stored data. But we cannot stress the database with all user's load because of limited scalability caused by the multi-tenant hosting environment. Therefore we use Windows Azure Storage to store cached data loaded from the database in the form that requires minimum processing by compute service to lower its load and allow better scalability of the frontend server. Disadvantage of this approach is a data duplication, but the overall costs for storage capacity in Windows Azure Storage are minimal. We should effectively combine Windows Azure Storage services like the Table Storage where we can store pre-processed data that can be addressed by unique key that is required for effective retrieval of stored data. For binary objects, like images and other resources that we need to distribute to client, we will use the Blob Storage. The client can download files directly from the Azure Storage without need to generate a load on our front-end servers in case files are stored in a public container in the storage. To increase the distribution capacity of this service we use the Content Delivery Network to distribute files that will not be changed often or we can bypass the inability to

invalidate a CDN cache when file changes by implementing a mechanism that generates new files with new names after each change.

To increase a performance of the storage system we can put the most often used data into the in-memory cache of the web server but we have to deal with the replication problem, because when we cache data that may change, each server can cache a different version and this may lead to inconsistent results when a load balancer lets client communicate each time with a different server in our service. Therefore we can use a Cache Service in Windows Azure that will host our in-memory cache in a farm of dedicated servers. The data retrieval from this cache is a way faster that from Azure Storage so we get performance benefits and at the same time there is not the problem with different versions of cached data. This service is the most expensive type of storage so we should put read-only data to the local cache and use this shared cache only for data that will change.

When we use all available storage technologies in the correct combination we can significantly increase scalability of the designed service and also optimize its response time and lower running its costs.

4.2 Compute Tier

Clients communicate with the designed service via a HTTP REST communication interface, which is hosted by a farm of web servers (does not matter if the Azure Cloud Service or the Azure Websites Service is used because this API can be hosted in both services and both mentioned offer vertical and horizontal scalability).

For resource intensive long running compute operations it is more efficient to use an asynchronous processing. This means that if the operation runs for a considerably longer time than other requests at the web server, it is better not to process the request but to put it into a processing queue. A different server running as a worker can load this operation from the shared queue and process it asynchronously.

This approach brings following benefits:

1. **Response time optimization.** The goal is to lower the overall load at front end servers to decrease the response time for client requests. Therefore we use more storage services as described earlier and we try not to process long-lasting requests that may exhaust compute resources so other requests will have to wait in an internal queue of the webserver for processing.
2. **Better load distribution.** Because the network load-balancer monitors only availability of the web server and not its current load (due to large amount of monitoring data and requirement for extremely fast request redirection) it can happen that load balancer will redirect a more resource expensive request to a small group of servers that will become overloaded. Other servers in the farm may be idle at the same time. The network load balancer cannot effectively distribute more complex compute operations without a knowledge of the servers load therefore we have to implement this mechanism on our own. At the design time we know what operations will be resource intensive so we can optimize them to run asynchronously on a different server.
3. **Recovery after error during processing.** It can happen that processing of the long-running operation fails due to an infrastructure failure or inner exception during processing. Windows Azure Storage Queue and Azure Service Bus Queue both implement two-phase message retrieval from the queue. Worker process loads a message and it becomes for a certain time invisible to others. If is the operation processed successfully the worker tells the queue to remove the message, but if the

worker fails the message with a task definition appears automatically back in the queue and can be processed by a different worker. Azure Service Bus Queue also supports moving messages to the specific queue after certain number of failures so they can by for example reviewed by a developer or an administrator.

This approach brings also some disadvantages:

1. **Asynchronous processing of the task.** Basic paradigm of the web and a stateless HTTP protocol is that a client requests data from a server and the server responds with data the client wants. The server cannot contact the client in case of an event at the server side. When we use asynchronous processing in the web environment there is problem that the server cannot send to the client results of the asynchronously processed operation, it can only send a confirmation that this operation was queued. When the operation is finished, there is no way how we can notify the client using a basic HTTP protocol. To resolve this issue we can use a modern approach of a WebSocket protocol. This protocol allows us to have an opened connection with every client so the server can deliver results of the asynchronously processed operation after its completion. It is also possible to off-load traffic to Windows Azure Storage by delivering to the client only an URL address where the results are stored. It depends mostly on the size of results. This behavior has to be implemented on our own and it may significantly increase an overall complexity of the service.
2. **Delay in task completion.** This approach may lead to longer overall time needed to process the results of less complex operations than in case of the synchronous processing because of a queue service delay, a wait time of the request in the queue and a notification delay.

Asynchronous processing of complex compute operations on different servers significantly improves overall service scalability.

4.3 Client Tier

The Client Tier is in this paper discussed very briefly because it is strongly dependent on a used technology and each technology has its specific rules for creating well-designed application architecture.

The chosen communication protocol for hosted web services is HTTP REST, which is supported by wide range of technologies used for development of client applications.

While designing the architecture of the client application it should be kept in a mind, that it should follow the same recommendations as mentioned for the service's architecture – clearly defined components with defined communication interfaces. When communicating with the server, the client should utilize a client-side caching when possible to decrease the service load.

5 Evaluation of implemented service using load tests

I have implemented a reference service according to described recommendations. This service was deployed to the Windows Azure cloud and some characteristics were benchmarked using the Performance Load Testing in Visual Studio. Load tests were running on my server in a datacenter to achieve a higher throughput of the Internet connection.

5.1 Test Case 1: Impact of local cache at service scalability

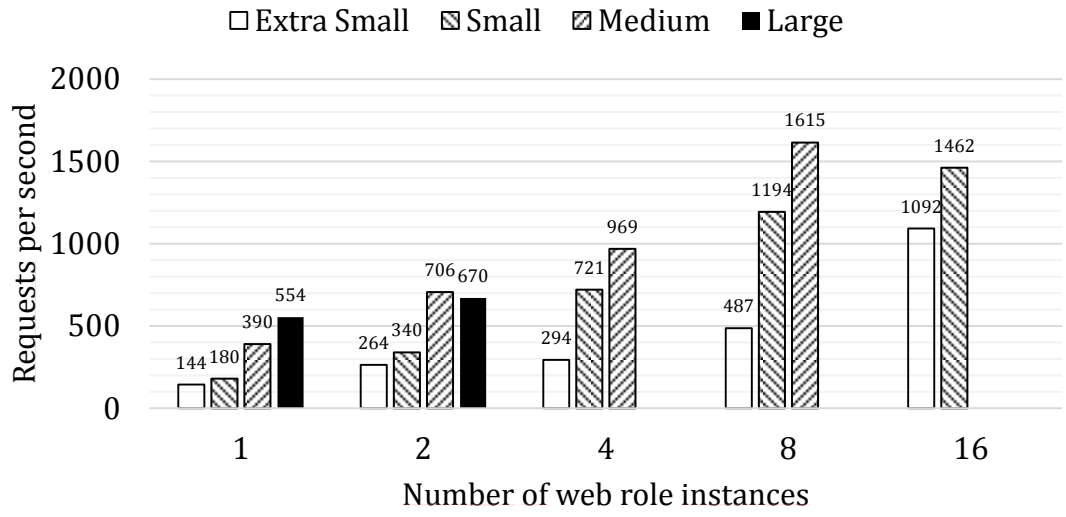
In this test two different implementations of a REST web service are evaluated. The first implementation every time loads data from an Azure SQL Database, the second implementation uses a local cache for data loaded from the database.

This service was tested in many configurations of the cloud environment to capture an impact of the vertical and horizontal scalability. Parameters of different virtual server configurations are described in Table 1.

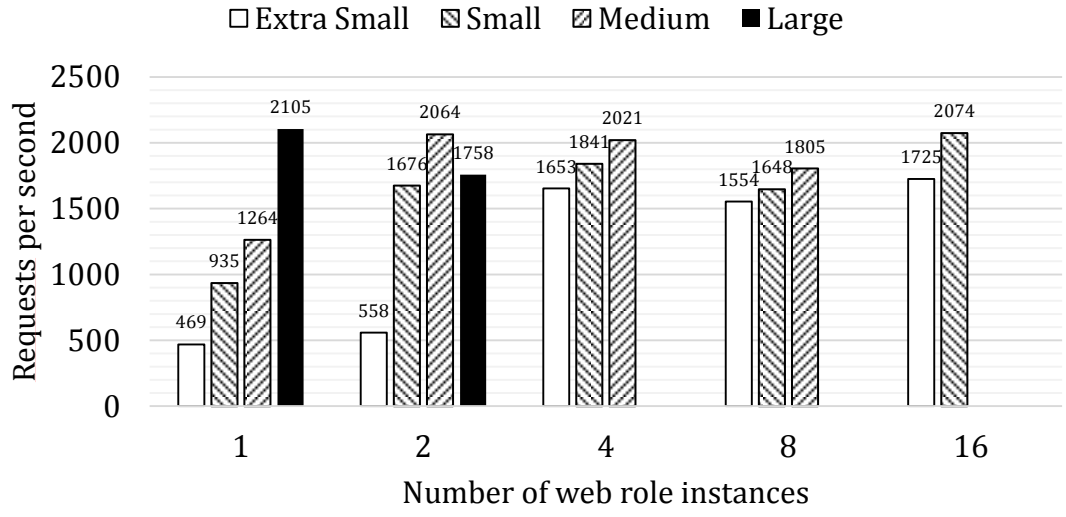
Compute Instance	CPU (GHz)	Memory	Storage	Bandwidth	Cost / hour
Extra Small (A0)	**1 (shared)**	**768 MB**	**20 GB**	**5 Mbps**	**0,02 $**
Small (A1)	**1.6**	**1.7 GB**	**225 GB**	**100 Mbps**	**0,08 $**
Medium (A2)	**2 x 1.6**	**3.5 GB**	**490 GB**	**200 Mbps**	**0,16 $**
Large (A3)	**4 x 1.6**	**7 GB**	**1000 GB**	**400 Mbps**	**0,32 $**
Extra Large (A4)	**8 x 1.6**	**14 GB**	**2040 GB**	**800 Mbps**	**0,64 $**
A6	**4 x 1.6**	**28 GB**	**1000 GB**	**1000 Mbps**	**0,90 $**
A7	**8 x 1.6**	**56 GB**	**240 GB**	**2000 Mbps**	**1,80 $**

Table 3: Available virtual machine configurations in Windows Azure

Load test results for the service without the local cache:



Load test results for the service with the local cache:



Included test results clearly shows the importance of more complex tiered storage, where version of the service with active in-memory cache can process 2-4 times more requests than version without cache.

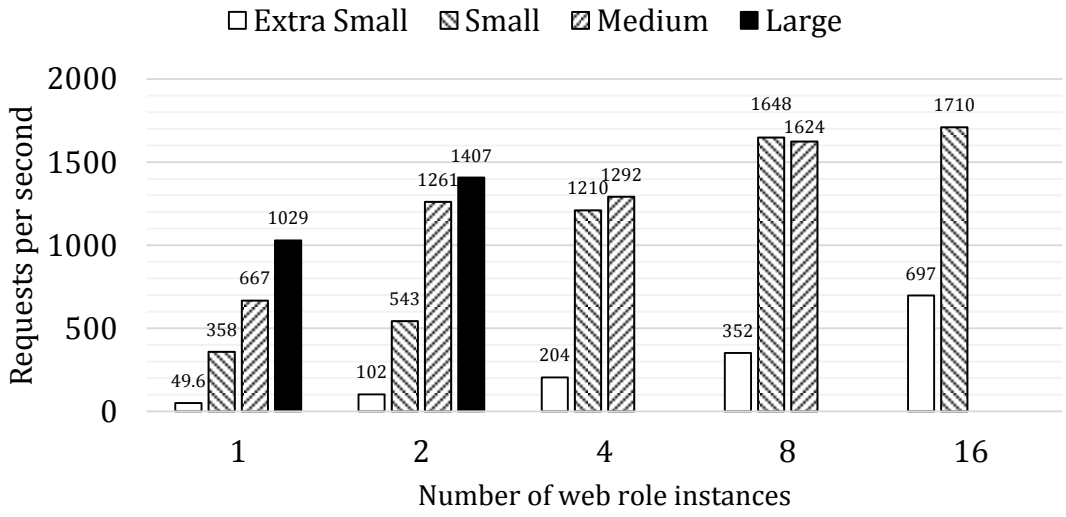
Note: I expect that this service is capable of processing even more requests, particularly in configurations of 8 and 16 instance, and in scenarios with medium and large instances. The performance bottleneck is probably the computer where the tests were running.

5.2 Test Case 2: Scalability of binary file delivery

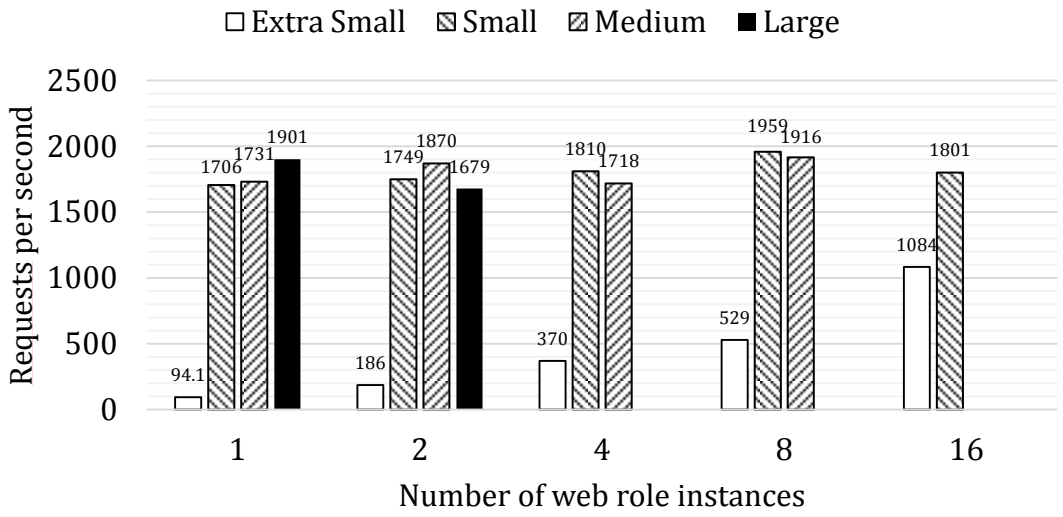
Significant part of the overall service traffic is caused by a transfer of images and other resources. Therefore I wanted to test which way of storing images is more effective and scalable.

The first implementation of a tested service loads images stored as binary streams in a database SQL Database. Storing binary files in a relational database is an approach that many current web services use. The second implementation stores data at a local drive of the web role. This approach is suitable only for resources that are part of an installation package of the application. The third approach tests performance of the Windows Azure Blob Storage. The fourth tests performance of the Windows Azure Blob Storage combined with the Content Delivery Network.

Results for the service implementation that loads image data from the database:



Results for the service implementation that loads image data from the local disk of the web role:



Windows Azure Blob Storage: **1845 requests/second**
Windows Azure Blob Storage + CDN: **1639 requests/second**

Test results demonstrate that currently used approach to store binary objects in a relational database is not as effective as use of the Windows Azure Blob Storage. When files are loaded from a local drive of the web role results are much better, but this scenario works only for files distributed together with an installation package. The Content Delivery Network has almost the same performance as the Windows Azure Storage.

6 Conclusion

This paper briefly introduces the most important services of the Windows Azure cloud platform and discusses their scalability and availability characteristics so the reader understands that current generation of web services which were built to run on a single web server loading data only from relational database cannot fully benefit from the cloud. To design architecture of highly available and scalable cloud service it is needed to take into account wide range of offered cloud services and discussed recommendations in Sections 3 and 4, which generally describe how to effectively use these services.

References

[1] MICROSOFT. *Windows Azure: Microsoft's Cloud Platform* [online]. 2013 [cit. 2013-11-10]. Dostupné z: http://www.windowsazure.com/en-us/

[2] BONDI, André B. Characteristics of scalability and their impact on performance. In: *Second International Workshop on Software and Performance, WOSP2000: proceedings : Ottawa, Canada, September 17-20, 2000* [online]. New York: Association for Computing Machinery, c2000 [cit. 2013-11-10]. ISBN 158113195x. Dostupné z: http://dl.acm.org/citation.cfm?id=350432

[3] CALDER, Brad et al. Windows Azure Storage: A Highly Available. In: *SOSP 2011 — Proceedings* [online]. 2011 [cit. 2013-05-24]. SOSP '11, October 23-26, 2011, Cascais, Portugal. Dostupné z: http://sigops.org/sosp/sosp11/current/2011-Cascais/printable/11-calder.pdf

[4] RICHARD N. TAYLOR, Richard N.Nenad Medvidović. *Software architecture: foundations, theory, and practice*. Hoboken, N.J: Wiley, 2010, s. 468-475. ISBN 9780470167748.

Performance Challenges, Current Bad Practices, and Hints in PaaS Cloud Application Design

David Gesvindr[*] [†]
Lab of Software Architectures and Inf. Systems
Masaryk University, Faculty of Informatics
Brno, Czech Republic
gesvindr@mail.muni.cz

Barbora Buhnova
Lab of Software Architectures and Inf. Systems
Masaryk University, Faculty of Informatics
Brno, Czech Republic
buhnova@mail.muni.cz

ABSTRACT

Cloud computing is becoming a popular approach to software application operation, utilizing on-demand network access to a pool of shared computing resources, and associated with many benefits including low-effort provisioning, rapid elasticity, maintenance cost reduction and pay-as-you-go billing model. However, application deployment in the cloud is not itself a guarantee of high performance, scalability, and related quality attributes, which may come as a surprise to many software engineers who detract from the importance of proper design of a cloud application, expecting that the cloud itself is the solution.

In this paper we analyze the issues and challenges associated with the design of a cloud application that has to be in compliance with given performance criteria, such as the throughput and response time. We also analyze the concerns related to other relevant quality criteria, including scalability, elasticity and availability. To support our findings, we demonstrate the identified performance effects of the examined design decisions on two case studies.

1. INTRODUCTION

Cloud computing can be defined as a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [14]. Thanks to these attributes, cloud computing attracts growing attention within both industry and academia as an alternative model of software system operation, associated with high system availability, scalability and performance. However, the potential of cloud computing to optimize these quality characteristics is rarely taken to its fullest due to yet unclear dependencies of these qualities on proper design of the cloud-based systems.

The crucial design phase in this respect is the architecture design of the system, when the basis for the performance-relevant quality characteristics is laid [3]. The software architect plays a key role, being active not only during architecture design, but often having responsibility for technical and technological leadership of the project, and overseeing development, testing and operation. During development of a software system that utilizes some form of cloud computing (referred to as *cloud application* within this paper), the role of a software architect in addition includes:

- Selection of the cloud operator based on provided cloud services, platform compatibility and service pricing.
- Feasibility study of the application's fitness for successful operation in the cloud environment, meeting all business and quality requirements.
- Comparison of the estimated operation costs in the cloud and on private infrastructure.
- Design of the software architecture taking into account the availability, scalability, response time, billing model and other related characteristics for every component operating in the cloud.

An integral part of the architecture design process is the selection of the suitable cloud service model or their effective combination. The cloud service models are the following:

- *Infrastructure as a Service (IaaS):* Model where the fundamental computing resources are provisioned to the customer, most often in form of a virtual server.
- *Platform as a Service (PaaS):* Developer platform capable of running cloud applications without control of the underlying cloud infrastructure (i.e. network, servers, operating systems, or storage). However, the customer has control over the deployed applications and possibly the configuration settings for the application-hosting environment.
- *Software as a Service (SaaS):* The cloud customer uses the provided software applications running on cloud infrastructure. The customer has no control over the underlying cloud infrastructure, and the application configuration is limited to a set of options made available by the provider.

[*]Professional awards and memberships: Microsoft Most Valuable Professional: Microsoft Azure, Microsoft Certified Solutions Expert: Data Platform, Windows User Group board member

[†]This publication was written with the support of the Specific University Research provided by MŠMT.

Although the SaaS model can also be interesting to the software architect when searching for third-party components to be integrated in their solution, performance challenges are rather associated with IaaS and PaaS, which give the architect more freedom in the design. Moreover, running IaaS virtual machines in the cloud is fairly similar to running virtual machines in an on-premise (i.e. private) environment. Therefore, PaaS cloud service model appears to be the most interesting from the performance-concerns perspective due to the lack of understanding of the built-in high-availability and scalability services, which are currently the key reason for many software architects to move their software applications to the cloud. These services include web hosting, storage and application services, which come with built-in high availability and easy vertical or horizontal scalability achieved through a complex service architecture that includes multiple virtual hosting servers and complex network topology, including load balancers and built-in monitoring. Thanks to these and other benefits, including the convenient configuration APIs provided by the cloud provider, Paas is becoming the choice number one for many software architects who are considering to employ cloud computing in their architecture design.

However, all the complexity associated with this new context puts significant demand on software architects excellent knowledge of cloud services and their performance-relevant effects on the design of their cloud applications, although these effects are not yet well understood and described in literature.

In this paper, we build upon three years of extensive experience with cloud application development, consultancy and evangelization, which helped us to observe many issues and challenges associated with cloud application design and development, especially when PaaS services are employed, which are not yet well described in literature. The contribution of this paper is the identification of the key issues and challenges associated with architecture design of a cloud application employing PaaS services that has to be in compliance with given performance criteria, including throughput and response time, and at the same time fulfilling other often conflicting quality characteristics including availability, scalability and elasticity. To increase the credibility of our findings, our observations are supported by two case studies, built upon the Microsoft Azure cloud platform.

The structure of this paper is as follows. After the overview of related work in Section 2, Section 3 presents the challenges related to the performance of cloud applications, and Section 4 discusses the relevant performance characteristics of cloud applications and best practices for their optimization in cloud-application development. Last, Section 5 concludes the paper.

2. RELATED WORK

The primary source of guidance on the design of cloud applications are typically the white papers published by cloud providers to help software architects and developers to correctly design scalable and highly available applications [8, 21, 12]. Although such papers provide the developers with very good high-level understanding of the domain, many specific issues (discussed in this paper) are left unexplained.

Another noticeable source of information on architecting scalable applications for the cloud originates from software architects and chief technology officers working in leading companies [15, 2, 19, 6] and providing the community of cloud practitioners with their personal experience in architecting large scale cloud applications, detailing their design mistakes and lessons learned from them. Moreover, positive examples of successful cloud application design include a number of large scale online services running typically in private rather than public cloud. These services include Wikipedia [1], Facebook [19] and Stack Overflow [6]. Although these texts are very valuable, their discussions are not put into the perspective of wide range of performance-related quality concerns studied in this paper.

Besides industrial guidance, we have surveyed numerous research papers, searching for a detailed view on specific performance challenges, their root causes and possible solutions. However, we have found that most of the cloud research we have identified focuses on the IaaS service model studying the principles of virtualization [18], effective virtual machine distribution [16], behavior of the underlying infrastructure [7] or providing tools for IaaS cloud simulation [4], while the PaaS services are not receiving much attention. The same is true for the papers connecting cloud research with software or service quality, studying cloud technologies and their quality attributes [10]. Also here, very little attention is paid to the PaaS service model and the combination of IaaS and PaaS within a single cloud application. Moreover, most of the cloud research has very narrow focus, which hinders the understanding of the big picture.

3. CHALLENGES

Our experience shows that one of the key reasons why software architects and developers face performance challenges when developing applications for the cloud, is that during the design, they tend to opt for services analogous to those they know from the fully controlled on-premise environment, and expect these to be the best fit for their purpose also in the world of cloud computing. By disregarding the key differences between the two platforms, this approach introduces many performance bottlenecks and scalability problems. This section lists the most intricate specifics and other challenges associated with the cloud environment that the developers and architects should be well aware of when building high-performance cloud applications.

Broad offer of platform services. Software architects and developers tend to use the cloud services whose equivalents are popular in the on-premise environment. Such a choice may lead to unnecessary performance bottlenecks, as the services with better performance characteristics may be missed. For instance, many developers decide to use relational database for storing all application data even though there is a certain subset of data not dependent on relational-database features (strong consistency, integrity enforcement, transaction processing) that can be comfortably stored in a high scalable NoSQL storage service with high throughput, short response time and lower operation costs.

Multi-tenant environment. Cloud service providers are investing into isolation of loads of different customers being hosted in the shared environment. Due to the complexity of PaaS services, the mitigation of side effects is in contrast with high density and efficient resource utilization in the cloud. Certain PaaS services offer service tiers with re-

served computing resources, where the services are hosted in a dedicated mode on isolated virtual machines with dedicated resources (CPU, memory, I/O) serving only the load of a single customer while retaining the benefits of PaaS (no required maintenance, simple configuration). However, even this operation mode cannot guarantee certain performance limits, due to other shared components—e.g. load balancers, network components and network lines in the data center—therefore noisy member effects may occur.

Cross-service communication latency. Architects sometimes neglect the fact that every call to a remote resource during request processing (e.g. database call, storage access) increases the overall response time of the application, and if the resource waits are improperly handled, due to imminent CPU blocking, an application throughput is also decreased. This effect is suppressed in an on-premise environment by the fact that related network resources are often collocated in the same rack connected to the same 1 Gbps or 10 Gbps switch, which leads to <1ms network latency where the major proportion of the delay is caused by the remote call processing itself. In the cloud environment, individual services may be spread across the entire data center, which leads to significant increase in network latency (10–100+ ms) in remote service calls, affecting the cloud application enormously.

Not all cloud services are designed to scale. In an on-premise environment, performance of hosted application is determined by available server resources, limited by server configuration that gives the share of the resources available to each application on the same server. While in IaaS environment the situation is similar, in PaaS environment the performance of the application and its guarantee is limited by selected services used to host the application and all other related resources the application uses to process its requests. PaaS services can be separated into two groups: The first has fixed performance and scalability characteristics, while the second allows increase in performance by choosing more expensive service tier or paying for more service instances processing the load simultaneously. For a software architect, it is necessary to take into account the current and future performance demands of the application to select the right PaaS services whose performance and scalability will not become an urge for late changes of cloud services and the architecture of an existing application.

Pay-as-you-go billing model. When using PaaS cloud for hosting applications, cost structure is different from running applications on proprietary hardware, where costs are given by acquisition costs (hardware + licenses) and maintenance costs (data center placement, maintenance) but are independent of actual load of the server (valid also for IaaS cloud), which in many cases leads to inefficient resource usage and lack of optimization. In case of PaaS, most of the services are billed in a complex way based on various usage statistics (per transaction, per reserved performance, per transferred data), therefore it is crucial to use the resources efficiently.

Undocumented behavior of the public cloud. One of the most intricate issues with the public cloud, not well stated and explained by cloud operators, is the inconstancy of service performance and transient faults. The former is caused by multitenancy of cloud platform services, when the same hardware resources are shared with multiple customers without sufficient load isolation. The latter is caused by the massive scale of cloud operation, when high availability is not achieved by highly reliable hardware, but by deployment on redundant computation and storage resources. In such a large scale, however, hardware outages occur on a daily basis, with many transient faults resulting from forced failover operations.

Poor availability and limited validity of design patterns and best practices. In case of on-premise environment, many well-documented design patterns and best-practices are available, which are however not validated in the public cloud. Due to the previously mentioned differences, many techniques are certainly not valid in the PaaS cloud and those that are valid, may have certain limitations that are not yet understood. For a software architect without deep practical experience with cloud application design, the use of common design techniques valid in the on-premise environment, may lead to significant quality decrease (namely in terms of performance) when used in the cloud.

Missing data and tools for performance modeling and prediction. Current techniques for performance modeling and prediction are designed mostly for the on-premise environment, relying on an assumption that the application is running as a process on a physical or virtual server with known computing capacity, memory and I/O operations. However, developers utilizing PaaS cloud services are in most cases shielded from an access to the infrastructure details. They only work with each service as a black box, thus not being able to access CPU, memory and I/O capacity in their models, which are in many cases highly dependent on very accurate metrics of the system.

Quality conflicts. Performance of a cloud application is only one out of many quality aspects that have to be thoroughly evaluated. Software architects designing cloud applications have to keep all the quality attributes in balance, which is a challenging task because of conflicts between some of the quality attributes. For example, poor performance of an application strongly dependent on a relational database can be improved by scaling-up the database which may lead to considerable increase in operation costs. At the same time, changing the relational database to a storage service with more suitable performance/cost ratio may be very hard due to ripple effects throughout the whole system architecture that has not been designed for this kind of change.

4. PERFORMANCE-RELATED QUALITY CONCERNS

The quality attributes discussed in this section, i.e. throughput, response time, scalability, elasticity, availability [13], provide the key performance-relevant metrics, whose analysis should be an integral part of capacity planning for a given cloud application. Their underestimation during the design phase may lead to significant quality degradation of the designed application.

This section details the impact of the cloud-related architectural design decisions on these quality attributes. As an

integral part of the discussion, we identify the dependencies of the given performance-relevant quality attributes on the type and configuration of different cloud services. Each of the subsections discusses the cloud specifics most relevant to an individual quality attribute. Best practices relevant to each of the quality attributes are summarized at the end of each subsection.

To collect the presented findings, we analyzed an extensive number of both complete and incomplete case studies and cloud experience reports (written and verbal). The case studies were related to projects developed both inside and outside of our team, some of them being published by other developers describing architecture-related issues with the implementation of cloud applications.

The first two subsections, dedicated to the throughput and response time, are accompanied with case studies, serving as the motivating examples for the discussion that follows. While the second case study reports on our own experience, the first case study elaborates on an experience of the Azure Customer Advisory Team, reported in [17] by Mark Russinovich (CTO of Microsoft Azure). In this text, we extend the original use-case with further explanation and clarification of the root causes for performance problems and the impact of the identified practices.

4.1 Throughput

Throughput can be defined as a measure of the amount of work that an application must perform in a unit of time [9]. Throughput is being quantified with a number of transactions, operations or requests that the system can handle per second or other time unit.

When working with this metric, it is crucial to distinguish between an average throughput and peak throughput because unclear specification of a throughput requirement may lead to decreased availability of the application because of the load congestion. The average throughput is defined as an average number of requests that the system needs to be able to process over a longer period of time. In contrast, the peak throughput requirement demands a guarantee of a number of processed requests in a very short time period, typically seconds.

4.1.1 Motivating example – Election tracking

The impact of cloud-relevant architectural design decisions on the application throughput can be very well demonstrated on the following application for online tracking of election results, employed by one of the US States [17], relying on the Microsoft Azure as its cloud platform[1]. Due to the performance degradation observed during September 2014 elections, the architecture of this application was consulted with the Azure Customer Advisory Team, which analyzed the application architecture and proposed architecture improvements that were validated during November 2014 elections [17]. This section presents the original and updated architecture of this application, as reported by Mark Russinovich (CTO of Microsoft Azure) in [17], and details the root causes for performance problems that go beyond the explanation in [17]. The observations are then generalized into the summary of practices for high-throughput cloud applications.

[1]Although the two case studies we employ are both based on Microsoft Azure and its services, all the findings and demonstrations are easily portable to other cloud platforms, such as Amazon, since we only refer to the services that have their equivalents in other cloud environments.

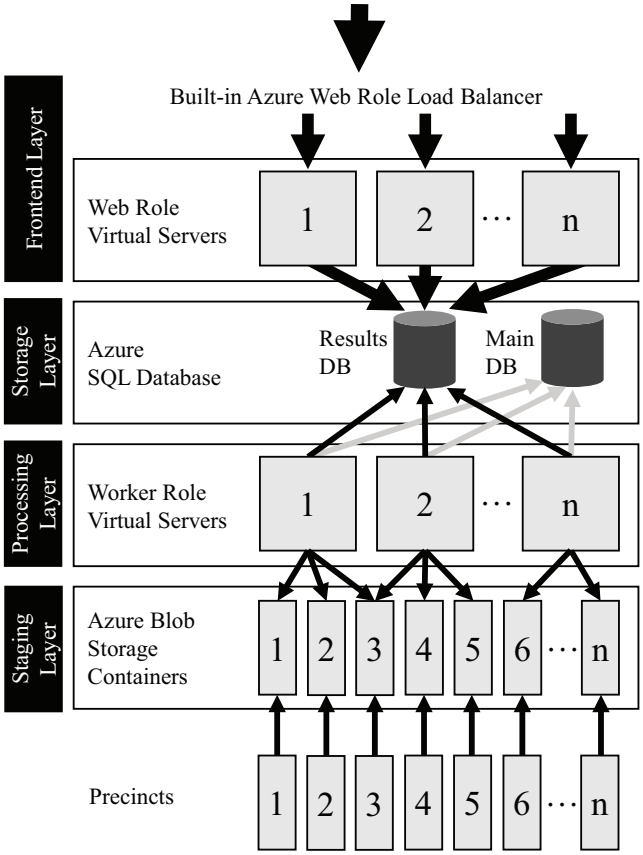


Figure 1: Original architecture of the system serving election results which illustrates an overloaded relational database limiting throughput of the entire application.

The initial architecture of the application (with performance problems) is depicted in Figure 1. Devices collecting voting results in precincts upload the results into Azure Blob Storage, each precinct having its own storage container. This storage service is highly available and scalable, and uses multiple storage containers to further increase its scalability. This architecture decision prevents the application from the overload of excessive incoming upload requests.

The uploaded data is processed asynchronously by a worker role, which is a stateless virtual machine with an application periodically checking the blob storage for new, yet unprocessed data. After downloading and processing data from the blob storage, the results are uploaded to the SQL Database. In the architecture design, two relational databases are used to separate the workload. The first database stores metadata and the second stores voting results. The front end is represented by web workers, stateless virtual servers running webservers with a front-end service deployed on them. The virtual servers are horizontally scaled and the incoming requests are load balanced.

To achieve redundancy in case of data center failure, the application is deployed in a second data center and the data is uploaded simultaneously on two different storage accounts hosted in two separate data centers. In case of application

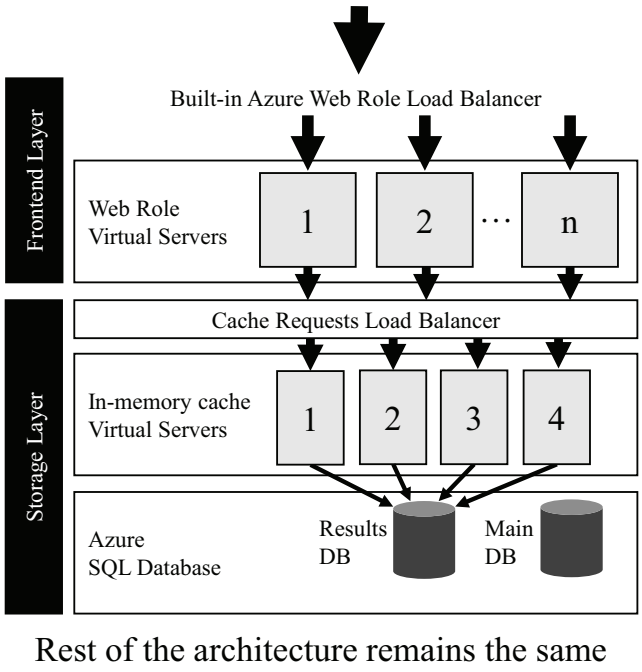


Figure 2: Updated architecture of the service including high throughput in-memory cache removing the load from the relational database.

unresponsiveness in the first data center, Azure Traffic Manager redirects the requests to the second data center, which is ready to handle the load. The advantage of this strategy is its straightforward implementation requiring only a simple modification of the upload process and deployment of the identical application to the secondary data center. A disadvantage is the missing load distribution, causing the application deployment in the secondary data center to be completely unused and billed until the failure of the primary one.

After detailed analysis of the client request processing, the performance bottleneck became clear. The analysis revealed that each incoming request needs 10 database calls for its processing. If we limit SQL Database throughput to 1 000 requests per second (i.e. the throughput limit of the premium SQL Database service provided by Microsoft in November 2014), the application throughput only reaches 100 requests per second, which is far below the expected peak load set as a requirement by the customer (1 667 requests per second in this case).

The observed bad practice that we would like to highlight is the architecture design relying on a single non-scalable relational database serving all application load. Our experience shows that this is a common practice of developers experienced in on-premise solutions, which utilize relational databases as the primary storage without considering its scalability, because for most on-premise scenarios the database server scales well enough. Another poor practice we identified on multiple other projects in this context is the over-usage of database abstraction techniques (O/R mappers etc.), which often lead to unnecessary overloading of the database because the developers easily lose track of the associated load costs. Due to the limited scalability of the database service in the cloud, the number and complexity

Time	Page Views/s	Cache Calls/s	Expected % of req. served with DB	% of req. served with cache
20:00:00-20:00:10	44 893	448 932	0.22%	35.64%
20:00:10-20:00:30	10 346	103 463	0.97%	100%
20:00:30-20:01:00	5 708	57 077	1.75%	100%
20:01:00-20:03:00	3 153	31 523	3.17%	100%
20:03:00-20:10:00	1 177	11 772	8.49%	100%
20:10:00-20:30:00	347	3 470	28.82%	100%

Table 1: Measured app. load and its capacity [17].

of database calls should be minimized by design.

To increase the scalability of a software component or service, Taylor et al. [20] recommend that system bottlenecks should be avoided by design. In this case the bottleneck is the Azure SQL database with limited throughput. First, we should question whether we can easily lower the database load by redesigning the application logic to minimize the number of unnecessary DB calls per user request. If it is not possible to increase the efficiency of DB usage, we should consider replacing the relational database with another storage service, which may not be feasible due to the required storage feature set.

The architecture of the application in the case study was updated to utilize a caching layer as depicted in Figure 2. Addition of caching servers that cache data loaded from the database increases the throughput of the server significantly, because unlike in the SQL Database, the throughput of one caching server was estimated to be 40 000 requests per second. Not to underestimate the peak load, 4 caching servers were deployed and the load was load balanced, which resulted in the total throughput of the caching layer of 160 000 requests per second, which is 160 times more than the throughput of the database.

Table 1 shows the measured behavior of the application with a caching layer deployed. For illustration, Table 1 includes the predicted behavior of the application with the original architecture. The peak load was in order of magnitude larger (44 893 requests/s), than the expected peak load (1 667 requests/s). Without the overprovisioned caching layer, the application would have become almost certainly unavailable.

4.1.2 *Summary of practices*

As one can see, there is a strong dependency of the application throughput on the architecture and the used platform services, which needs to be taken into account by the software architect when designing a cloud application.

Very dangerous misbelief is that any application hosted in the cloud comes with high throughput which is limited only by the amount of money we are willing to pay for resources in the cloud. The opposite is true. Selection of even a single poorly performing cloud-platform service is likely to result in significant drop in application's throughput.

The main takeaways of this section can be summarized into the following best practices:

- *Be familiar with the real throughput of your cloud services* – Every cloud service available within the cloud platform has its throughput limit. The problem is that the throughput limits of cloud services are not always well documented, therefore some effort is necessary to collect the throughput metrics and identify the suitability of the cloud service based on them.
- *Be aware of cloud services with hard throughput limits* – Some cloud services are poorly scalable and have strict throughput limits given by their inner design. The developer should be aware of these services, because when the limit is reached, the only way to increase the throughput of the developed cloud application is the costly redesign connected to the elimination of the limiting cloud service from the architecture.
- *Minimize the number of service calls* – The number of cloud service calls should be minimized by design of the application and later by its implementation. Especially when using PaaS with the pay-as-you-go billing model, minimization of service calls not only increases the throughput and decreases the response time but also decreases the operation costs. Although it is not always possible to minimize the number of dependency calls, the cloud applications should be anyway designed with this goal in mind.
- *Use batch processing* – Queue or storage services allows processing of requests in batches. Instead of sending individual messages to the queue or rows to the NoSQL storage, requests should be grouped into a batch of operations and then submitted as a single operation. The surprising fact about the effect of batch processing is that it can increase the throughput of the service up to 50 times.

4.2 Response time

Response time is a measure of the latency an application exhibits in processing a business transaction [9]. In the context of a cloud application, it is important to distinguish between network latency and request processing time. The former is a delay in the communication between the client and the data center, or to be more precise the virtual server where the application is running[2]. The latter is influenced by the application architecture and response times of all the (cloud) services that are invoked during request processing.

4.2.1 Motivating example – Private social network

One of the recent projects we were involved in is a private social network intended to support education process and use of tablet devices at elementary schools. This application is designed for high throughput by using highly scalable storage services together with techniques to minimize web server load. The load is minimized thanks to pre-computed request results by an asynchronous worker process running on a different set of virtual machines, which can be horizontally scaled so that most of incoming client requests that load data are served directly from NoSQL database without

[2]Even in the case of a PaaS service, the request is processed by a virtual server that cannot be managed by the developer.

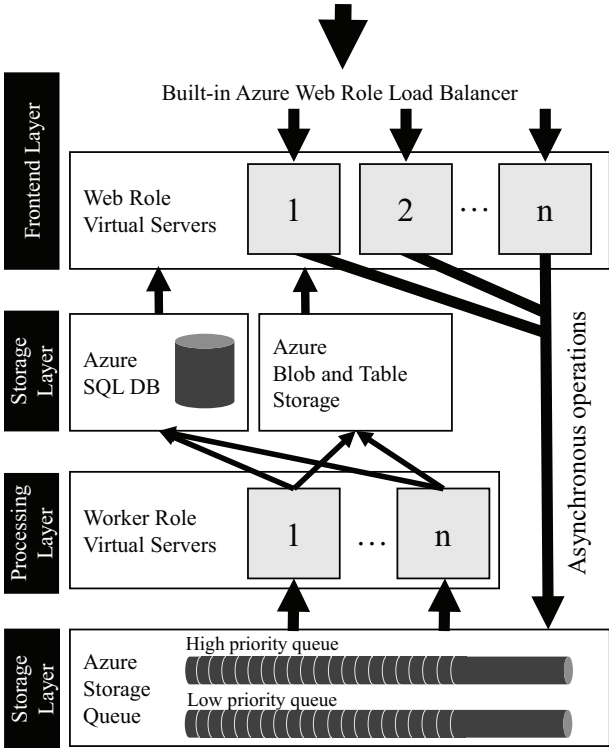


Figure 3: Software architecture of implemented application with an asynchronous request processing.

complex processing. The write requests, on the other hand, are persisted directly to a highly scalable queue. It is important to realize that write request (e.g. new message on the wall, new comment, submitted homework) does not have to be processed immediately. The client application written in JavaScript displays new records on the wall as soon as they are stored in the queue (which takes under 500 ms) so the user experience is not affected by delayed processing by the worker process.

When we implemented the initial version of this application, we found out that the worker process that generates the cache in NoSQL database is processing messages from the queue with a very poor response time, exceeding 5 seconds per request, which is far below our expectations.

To detect the root cause, we have implemented a basic monitoring system, which stores execution times of every processed message. This helped us to identify the messages with poor processing times. In the system there are two major types of requests processed by the worker - the first stores data from a client to the primary storage which is a relational database, the second loads data from the database and pre-computes results for the NoSQL database, which handles the majority of user load.

While the requests of the first type had the response time below 500 ms, the requests updating the NoSQL database were significantly slower.

To identify the exact delays in request processing, we instrumented the code with a set of stopwatches, measuring the exact execution times of suspected blocks of code. Using this technique we identified three major sources of processing delay, which are the following.

- *Excessive database calls* – During request processing we called the database approximately 10 times (with an average response in 200 ms) to load all data necessary for cache entries generation. We discovered that merging some database calls together significantly improves the performance due to avoided round trip delay between virtual server and the database. Other technique we used to minimize the number of database calls was an extension of the main entity (e.g. wall record) loaded from the database with flag columns (e.g. marking if the record has some attached files) identifying the operations that can be skipped (e.g. loading an empty list of files from the database).
- *Inefficient algorithms and redundant computation* – Because we cache data in a pre-computed form specific for each user (simple query to NoSQL database with fast response time) we identified an inefficient algorithm that generated cache entries, which was the source of another delay in processing. The key problem was that we repeated a complex data transformation for every user although these were user-independent and hence could have been done once and the result cloned for every user.
- *Partially synchronous write to the NoSQL database* – Although we used asynchronous processing when writing multiple records to the same table, we employed synchronous processing when writing to different tables (i.e. waiting till the writes to one table finish before we started to write to another table), which was another source of a significant delay that could have been removed.

4.2.2 Summary of practices

Following techniques summarize the identified best practices that can be imployed to improve the response time of a cloud application:

- *Use low response-time services* – Cloud platforms offer various services with similar functionality and different response times, which can be used in a cloud application. In design time, the biggest issue is that the response time of a cloud service is not fixed, but is dependent on the type of request and the amount of data processed or returned by the service. It is therefore recommended that the developers invest the effort into the measurement of response times of the relevant cloud services, so that these can be used as a basic guidance in the application design.
- *Minimize the number of service calls* – Although this technique has already been described in Section 4.1, it is important to emphasize its importance in the context of response time optimization too.
- *Merge simple calls* – When working with the same resource, examine the possibility to merge multiple simple calls to a more complex one, thus reducing the number of round trips between services, which in case of PaaS services and the previously mentioned cross data center communication latency is likely to lead to a significant performance improvement. For instance, multiple database calls loading scalar values from a database can be encapsulated into a single stored procedure, which can load the data from the database using multiple queries processed internally on the database server, but all the results would be send to the client with a single call. This stored procedure may have slightly higher response time, but in comparison to multiple database calls will certainly imply significant performance improvement.
- *Optimize service calls* – When calling cloud services, it is important to use efficient data access methods and load only the necessary data (e.g. use projection to return only the necessary columns from the database table, use server-side filtering to limit the number of rows which are returned to the client, etc.).
- *Call services in parallel* – When a cloud application makes multiple service calls, such calls can be handled in a synchronous manner - i.e. first the call is initiated, CPU thread is blocked until the response arrives, results are processed, and another call is made. This approach, however, significantly prolongs the response time, which is then computed as a sum of response times of all the calls plus the time for processing the results. If supported by the programming language, it is much more efficient to use the asynchronous approach, when multiple service calls are initiated at the same time, the CPU thread is released to be used for different purposes and when the responses arrive, the computation is restored and data is processed. This approach is not as straightforward as it seems, because not all requests can be processed concurrently due to dependencies between them, but when possible, requests should be processed in parallel.

4.3 Scalability and elasticity

In this section, we discuss the scalability and elasticity, which are often mistaken and sometimes also confused with other performance-relevant attributes. Hence, we first shortly clarify their meaning. The core performance characteristics, namely the *throughput* and *response time*, specify the application behavior for a static instance of cloud environment configuration. On the other hand, *scalability* of an application defines how the application is expected to behave with increasing load and dynamic server resources, and whether it is able to effectively utilize newly provided resources. *Elasticity* of the platform describes how fast and accurately is the environment able to adapt to workload changes by changing the amount of provisioned resources [11].

4.3.1 Scalability

In case of cloud applications, it is critically important to design the applications to be scalable. Otherwise the applications cannot effectively use the potentially unlimited amount of processing resources that the cloud platform offers.

From the technical point of view, scalability of an application can be achieved in a vertical or horizontal manner. In the case of vertical scalability, the cloud application should be able to utilize increasing computational resources of a single node within the cloud platform. This type of scalability is equivalent for instance to an upgrade of an allocated virtual server, upgrading from 4 cores to 8 cores, or by upgrading the frequency of the processor. In case of a web

application, the vertical scalability is usually not an issue, because a web server processes incoming requests in isolation, and thus can parallelize the processing utilizing more cores.

In case of a horizontal scalability, new computational nodes (servers) are made available within the cloud platform and the aim is to distribute the load among them. The load is being distributed with the help of a load balancer being placed in front of the set of computational nodes, which accepts client requests and forwards them to the computational nodes for processing based on a specific algorithm (most commonly the round-robin).

4.3.2 Summary of practices

- *Design the application for stateless request processing* – Horizontal scalability requires a specific software architecture using stateless request processing. Due to the lack of guarantee which server will process a future request coming from the same client, the application cannot rely on storing the state information about the processed request internally on the server. Because of that, the session state needs to be stored in a scalable central repository shared by all the servers.

- *Distribute the load* – Despite the fact that cloud providers offer both vertical and horizontal scalability for most of the offered services, vertical scalability imposes resource limits restricting the overall application scalability, due to hardware limits of the used servers. At the same time, high-performance servers are disadvantageous due to high costs per resource. Therefore it is recommended not to rely too much on the vertical scalability, and rather design the application for horizontal scalability, which allows the allocation of potentially unlimited number of service instances sharing the load.

- *Use scalable cloud services* – When identifying the cloud service fitting the requirements of a cloud application, the architect should keep in mind that the performance requirements may change and hence the scalability of the service should be well understood before the service is utilized. This is especially important because not all cloud services are designed to scale, and if this becomes a problem, costly redesign of the application may be the only solution.

- *Be aware of service scalability mechanics* – When using a PaaS service it is necessary to understand how it is designed to scale, thus how to efficiently use such a service without introducing artificial scalability limits caused by improper use of the service. For instance, Azure Table Storage is a NoSQL database that internally partitions data across storage servers [5]. Each partition has limited throughput, but when the application distributes the load properly across multiple partitions, the service scales without any limits.

4.3.3 Elasticity

Elasticity is a relatively new quality attribute closely related to the cloud environment. From the software architecture point of the view, elasticity is a quality attribute of the cloud platform, not an attribute of the cloud application, although application scalability is a necessary precondition for benefiting from the cloud platform elasticity. The key aspects of the elasticity adaptation process are [11]:

- **Speed** – The speed of scaling up is defined as the time it takes to switch from an underprovisioned state to an optimal or overprovisioned state. The speed of scaling down is defined as the time needed to switch from an overprovisioned state to an optimal or underprovisioned state.

- **Precision** – The precision of scaling is defined as the absolute deviation of the current amount of allocated resources from the actual resource demand.

4.3.4 Summary of practices

The key best practices for optimal elasticity are the following:

- *Use horizontal scalability* – An effective way to allocate new resources in case of an underprovisioned state and release resources in case of an overprovisioned state is by provision and removal of computational nodes. This process does not affect the rest of the deployed application, because only at the moment when the new node is ready to process an incoming request, the load balancer is configured to redirect requests also on this node. In case of vertical scalability, it is impossible to change such a substantial property of virtual machine as a number of CPU cores without node restart, which in case of single deployed node would make the application unavailable to clients.

- *Collect and process performance metrics for decision making* – It is necessary to identify a set of metrics, signalizing when the application is underprovisioned and when it is overprovisioned. These metrics can be basic performance metrics provided by the cloud service—e.g. CPU utilization, and network bandwidth utilization—or more complex metrics measured inside the cloud application—e.g. length of the requests queue.

- *Save effort by using auto-scale services* – Some of the cloud services come with an auto-scale option or an additional service, which measures specified metrics and based on defined rules and thresholds scales the application up or down. This auto-scale option can be also used for time-based rules, when peek and off-peak hours are defined for the application and a number of allocated nodes can be reduced in an off-peak hours which can lead to 30-50% operation cost reduction, without an increase of development costs, because this feature can be often easily managed via the management interface of the cloud provider.

- *Allocate resources before the peak load passes* – When scaling the application based on the measured load, it is necessary to consider how long it takes to allocate new service resources. For instance, in the case of Azure Cloud Service, it may take around 5 minutes until the new machine is provisioned and the application deployed on it. In that time, the peak load may already be over. On the other hand, some resources may be provided almost instantly, such as the Azure Web

Applications, which have very short provisioning time (tens of seconds), likely because of idle pre-provisioned nodes. When the application load fluctuates a lot, the provisioning process should be rather fast to match the current load conditions with a minimum delay.

Because of the close interconnection, scalability and elasticity should be analyzed together, best after the throughput and response time of individual application components are understood, so that the expected load can be estimated. Within the two, it is preferable to first analyze the scalability of the cloud application based on the discussed aspects, and then analyze the elasticity of the employed cloud services, identify load measures used for service load assessment, and define the rules for scaling up and scaling down.

4.4 Availability

Availability is defined as a percentage of the time when the system is operational [9]. In the cloud environment, every IaaS, PaaS and SaaS service offered by a trustworthy cloud operator has strictly defined availability in its service level agreement (SLA). The availability guarantee often exceeds 99.9%, being commonly 99.95% or 99.99%, which in case of 99.99% means at most 4 minutes of service interruption within a month. However, the big picture is much more complex.

First, the guarantee of high availability is sometimes bound to additional requirements, such as that the service must be running on at least two computational nodes, which may lead to doubling the operation costs and may require architectural changes in the application to support request load balancing among the two nodes, or switching between the active and passive node, which may significantly impact performance characteristics of the application.

Second, as mentioned in Section 3, paragraph *Undocumented behavior of the public cloud*, one of the big issues with the cloud is the high number of transient faults caused by failover operations at the platform level, happening out of our control when hardware or software component of the cloud platform fails. Since these errors last only a few seconds, a large number of failures (that can be further propagated) can occur during a month without high-availability guarantee violation.

Third, to be able to prevent global outage of the application caused by the data center failure (e.g. due to natural disaster or global provider outage), it is not sufficient to rely on cloud provider failover services. It is necessary to architect the application to efficiently operate across multiple data centers, which is a challenging task.

Last, even fully available cloud infrastructure does not guarantee that the designed cloud application itself will be available, because its availability strongly depends on: (1) computational, storage, and other third party services, including custom software services, and (2) the code of the application itself, which may trigger an internal error, which locks the application in a faulty state, making it unavailable.

4.4.1 Summary of practices

First of all, it is necessary to realize that running an application in the cloud does not automatically provide the application with the impressive 99.99% availability of the platform. To get closer to this guarantee, the cloud application must be designed to be capable of dealing with transient faults and outage of employed services. Moreover, each approach to increase the availability (including those listed here) should be put into the perspective of its performance side-effects.

- *Use services with built-in high availability* – It is beneficial to take advantage of built-in high availability of many PaaS services, which often comes at no extra cost, instead of building proprietary fail-over cluster in the IaaS environment, which is nontrivial to configure and operate.

- *Handle transient faults* – Due to transient faults, which are occurring in the cloud on a daily basis, it is necessary to implement retry logic, which in case of transient faults automatically triggers a defined retry policy and repeats the request multiple times before raising an exception signaling that the service request failed. The lack of retry logic will directly affect user experience, due to randomly occurring server errors, while using the application. On the other hand, the implementation of the retry logic in case of transient fault significantly and randomly increases the response time. We are currently not aware of any technique to effectively mitigate the spikes in response times, which are randomly occurring.

- *Consider service recovery scenarios* – Together with the availability defined in the SLA, it is important to consider various recovery scenarios, which shall bring the application back to operation in case of outage caused by cloud provider or the application itself, caused for instance by data corruption or faulty data deletion. To recover the application from such a failure, it is necessary to plan periodical backups or utilize built-in support for data restoration if such an option is available within the platform.

- *Design to run in a restricted mode* – To increase the availability, the application may be designed in a way that it detects outages of its components and employed cloud services and blocks related functionality or switches to read-only mode to prevent complete outage.

5. CONCLUSION

This paper examines the performance-related issues and challenges associated with the development of applications relying on some form of cloud computing. In the paper, we focus on individual performance-relevant quality characteristics of cloud applications—i.e. the throughput, response time, scalability, elasticity and availability—and for each of them we discuss the issues and best practices associated with the optimization of the developed cloud application.

Along the text, it became clear that performance engineering of cloud applications should be a continuous process starting in an early design phase of the project when the application performance requirements should be carefully planned and analyzed based on the architecture of the designed application and its services. During application development and later its operation, performance metrics should be collected and validated with a set of tests so that any deviation from the requirements is identified and fixed as early as possible.

Finally, it would be very beneficial to have access to specialized tools and design patterns that can help the developers and software architects in the situations described in this paper, which is one of the directions we would like to contribute to in the future.

6. REFERENCES

[1] Wikimedia servers. =https://meta.wikimedia.org/wiki/Wikimedia_servers, 2015. Accessed: 2015-10-25.

[2] S. Anand. Netflix's cloud data architecture. InfoQ, 2011.

[3] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley, 2003.

[4] R. Buyya, R. Ranjan, and R. Calheiros. Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities. In *High Performance Computing Simulation, 2009. HPCS '09. International Conference on*, pages 1–11, June 2009.

[5] B. Calder et al. Windows azure storage: A highly available cloud storage service with strong consistency. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 143–157, New York, NY, USA, 2011. ACM.

[6] M. Cecconi. The architecture of stack overflow. code.talks, 2011.

[7] N. M. K. Chowdhury and R. Boutaba. A survey of network virtualization. *Computer Networks*, 54(5):862 – 876, 2010.

[8] T. Ganesh. Designing a scalable architecture for the cloud @ONLINE, June 2012.

[9] I. Gorton. *Essential software architecture*. Springer Science & Business Media, 2006.

[10] M. Hauck, M. Huber, M. Klems, S. Kounev, J. Müller-Quade, A. Pretschner, R. Reussner, and S. Tai. Challenges and opportunities of cloud computing. *Karlsruhe Reports in Informatics*, 19, 2010.

[11] N. R. Herbst, S. Kounev, and R. Reussner. Elasticity in cloud computing: What it is, and what it is not. In *ICAC*, pages 23–27, 2013.

[12] B. Jimerson. Software architecture for high availability in the cloud. *Oracle Technology Network*, June 2012.

[13] S. Lehrig, H. Eikerling, and S. Becker. Scalability, elasticity, and efficiency in cloud computing: A systematic literature review of definitions and metrics. In *Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures*, QoSA '15, pages 83–92, New York, NY, USA, 2015. ACM.

[14] P. Mell and T. Grance. The nist definition of cloud computing. 2011.

[15] Y. Nelapati and M. Weiner. Scaling pinterest. InfoQ, 2013.

[16] J. T. Piao and J. Yan. A network-aware virtual machine placement and migration approach in cloud computing. In *Grid and Cooperative Computing (GCC), 2010 9th International Conference on*, pages 87–92, Nov 2010.

[17] M. Russinovich. Lessons from scale. =http://channel9.msdn.com/Events/TechEd/Europe/2014/CDP-B337, 2014. Accessed: 2015-10-25.

[18] J. Sahoo, S. Mohapatra, and R. Lath. Virtualization: A survey on concepts, taxonomy and associated security issues. In *Computer and Network Technology (ICCNT), 2010 Second International Conference on*, pages 222–226, April 2010.

[19] A. Sobel. Scaling the social graph: Infrastructure at facebook. InfoQ, 2011.

[20] R. N. Taylor, N. Medvidovic, and E. M. Dashofy. *Software architecture: foundations, theory, and practice*. Wiley Publishing, 2009.

[21] J. Varia. Architecting for the cloud: Best practices. *Amazon Web Services*, 2010.

Architectural Tactics for the Design of Efficient PaaS Cloud Applications

David Gesvindr [1]
Lab of Software Architectures and Inf. Systems
Masaryk University, Faculty of Informatics
Brno, Czech Republic
Email: gesvindr@mail.muni.cz

Barbora Buhnova
Lab of Software Architectures and Inf. Systems
Masaryk University, Faculty of Informatics
Brno, Czech Republic
Email: buhnova@mail.muni.cz

***Abstract*—Platform as a Service (PaaS) is one of cloud service models associated with effective system operation in the presence of a rich set of platform services. Due to the complexity of the PaaS service model, software architects need to face various challenges when designing cloud applications that have to meet certain quality guarantees.**

In this paper we report on our experience with the design and implementation of highly scalable PaaS cloud applications, which resulted into deep understanding of the weaknesses and benefits of this service model, and helped us to identify and evaluate a number of architectural tactics for PaaS cloud applications. Besides the identification and evaluation of the tactics, this paper studies the mutual relationships and the impact of the tactics on various aspects of application's quality. The findings are demonstrated on a real-world case study of a complex private social network system, based on the Microsoft Azure cloud.

I. Introduction

Cloud computing has became a popular approach to software application operation, utilizing on-demand network access to a pool of shared computing resources, and associated with many benefits including low-effort provisioning, rapid elasticity, maintenance cost reduction and pay-as-you-go billing model [1]. However, application deployment in the cloud is not itself a guarantee of high performance, availability, and other quality attributes, which may come as a surprise to many software engineers who detract from the importance of proper architecture design of a cloud application, although the architecture design plays a key role in the quality of the application.

An especially challenging context for software architects is the *Platform as a Service (PaaS)* cloud model, which relies on a platform providing the developers with an extensive set of services that should be well integrated into the application.

In this paper, we build upon experience with cloud application development, consultancy and evangelization, which helped us to observe many challenges associated with cloud application design and development, especially when PaaS services are employed. The contribution of this paper is the identification and evaluation of architectural tactics for the design and development of efficient PaaS cloud applications, together with their suitability in different situations and benefits of their combination with respect to the targeted application's quality.

The discussion of the tactics is related to one of the biggest real-world projects we have participated in, which involved the architecture design of a private social network supporting education on elementary schools in the Czech republic. Our role as the consultant was the architecture design of the cloud backend under strict performance constraints.

The paper is structured as follows. After the discussion of related work in Section II and outline of the case study system architecture in Section III, Sections IV and V are dedicated to the presentation and evaluation of seven architectural tactis for PaaS cloud application design. We conclude in Section VI.

II. Related Work

When designing a software architecture, it is a common good practice to use existing design patterns [2], [3], which are however not specifically designed for cloud applications. Therefore new catalogs of patterns relevant for the cloud are emerging [4], [5], which are however not yet supported with demonstration in real world settings, which could reveal the hints for efficient employment and combination of these patterns. One of the most relevant sources of cloud patterns is [6], which provides the most detailed pattern description and outlines their impacts. This source however lacks the discussion of the effective combination of the patterns and their relation to the cloud-application quality attributes. Explanation of software quality attributes generally valid also for the cloud is for instance in [7], [8].

Besides industrial guidance, we have surveyed numerous research papers, searching for a detailed view on specific performance challenges, their root causes and possible solutions. However, we have found that most of the cloud research we have identified focuses on the IaaS service model studying the principles of virtualization [9], effective virtual machine distribution [10], behavior of the underlying infrastructure [11] or providing tools for IaaS cloud simulation [12], while the PaaS services are not receiving much attention.

Other relevant source of guidance on the design of cloud applications are typically the white papers published by cloud providers to help software architects and developers to correctly design scalable and highly available applications [13],

1 Professional awards and memberships: Microsoft Most Valuable Professional: Microsoft Azure, Microsoft Certified Solutions Expert: Data Platform, Windows User Group board member

[14], [15]. Although such papers provide the developers with very good high-level understanding of the domain, many specific issues addressed by this paper are left unexplained.

Another noticeable source of information on architecting scalable applications for the cloud originates from software architects and chief technology officers working in leading companies [16], [17], [18], [19] and providing the community of cloud practitioners with their personal experience in architecting large scale cloud applications, detailing their design mistakes and lessons learned from them. Moreover, positive examples of successful cloud application design include a number of large scale online services running typically in private rather than public cloud. These services include Wikipedia [20], Facebook [18] and Stack Overflow [19]. Although these texts are very valuable, their discussions are not put into the perspective of wide range of performance-related quality concerns studied in this paper.

III. Solution architecture overview

For the demonstration of the addressed tactics and their quality-related effects, we have chosen a system designed as a private social network supporting education on elementary schools in the Czech republic. The target users of this application are the students and teachers of all elementary schools in the Czech republic, which based on the Czech Statistical Office [21] involves 4 106 schools, 43 259 classes and 854 137 students. Our primary responsibility was to design the backend service in such a way that it can sustain the load, which means:

- The service can fluently scale together with growing number of users and generated load
- The used architecture and technologies do not impose any insurmountable obstacle in service scalability that would later require complete service redesign (minor performance tuning updates are acceptable)
- The service has effective operation costs which scale fluently with the amount of active users

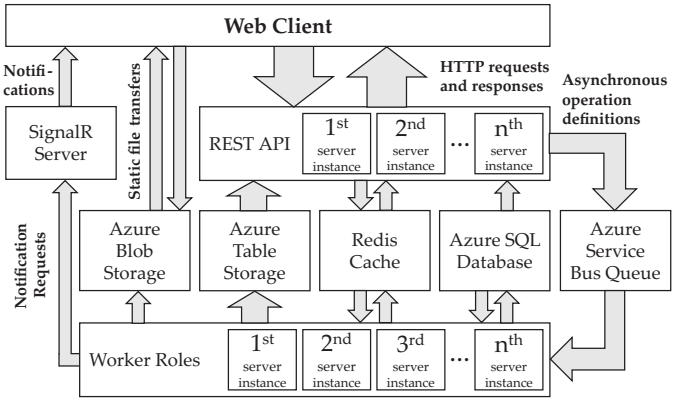


Fig. 1. Architecture of the designed solution.

The solution architecture of the designed cloud application, based on Microsoft Azure cloud, which resulted from the design decisions elaborated in this paper, is depicted in *Figure 1*. The architecture consists of the following components:

a) Web client: The client for this application is a single-page JavaScript application running in a web browser which takes advantage of the responsive design to be optimized for the use on high variety of devices including mobile and touch-enabled devices. The web client communicates with a REST API, opens a web socket notification channel using ASP.NET signalR and downloads files from Azure Storage.

b) REST API: REST API is the primary communication interface serving a majority of client requests. All data for the client excluding binary files are loaded using this API which is implemented as a PHP application hosted as Web Application in Azure App Service. This component loads data from Azure Table Storage, Redis Cache and Azure SQL Database.

c) ASP.NET SignalR Server: A notification server which delivers notifications to connected users using web sockets. Notifications are generated by workers and delivered to this server via an internal REST API.

d) Storage Layer: Data of the application is stored in multiple PaaS storage services as explained later, which includes both relational database (Azure SQL Database) and multiple NoSQL databases (Azure Table Storage, Redis Cache).

e) Worker Role: Worker role is an application which runs in an infinite loop and loads messages from the Azure Service Bus Queue where the REST API stores serialized commands with data modifications. This application mediates writes to the relational database, updates data in the NoSQL databases and generates real-time notifications.

IV. Storage and Data Access Tactics

As there is a rich set of storage services provided by PaaS cloud provides, it was necessary to first comprehensively evaluate their functionality, performance, availability and associated costs. Since the case study presented in this paper is based on Microsoft Azure cloud, *Table I* presents the results of our comparison for this provider. The comparison of storage services for other cloud providers is from the perspective of their features and qualities very similar.

Based on the comparison in *Table I*, it is clear that there is no universal storage service that would outperform all other services in terms of rich query options, scalability, high throughput, low response time, and operation costs. Therefore, to create a high-throughput storage layer, it is necessary to effectively combine multiple storage services to design the software architecture that meets desired quality criteria. In this section, we examine relevant existing architectural tactics and formulate their analogies for PaaS cloud, with respect to this goal.

A. Materialized View

1) **Motivation***:* One of the common cloud application issues is the architecture design relying on a single non-scalable relational database serving all application load, because for most on-premise scenarios the database server scales well enough. Another poor practice we identified on multiple projects in this context is the over-usage of database

TABLE I
COMPARISON OF AVAILABLE STORAGE SERVICES IN MICROSOFT AZURE

	Azure SQL Database	**Azure Table Storage**	**DocumentDB**	**Redis Cache**
Data Model	*Relational*	*Persisted Key-Value*	*Document*	*In-memory Key-Value*
Format of stored data	**Rows** stored in a table with **fixed schema** with one or multiple columns	**Rows** stored in a table **without fixed schema**, where each row can have different columns.	**JSON object**	**Key-Value pair**, where value is type of string or set of strings
Integrity enforcement	**Complex** (Primary key constraint, unique key constraint, data type of the column, foreign key constraint, check constraint)	**Limited** (partition and row key must be together unique)	**Complex** (JSON object validation rules written in JavaScript)	**Limited** (uniqueness of the key)
Query support	**Complex**	**Limited** (using partition and row keys)	**Complex**	**Limited** (using key)
Availability	**Very High** (99.99%)	**High** (99.9%)	**High** (99.95%)	**High** (99.95%)
Scalability	**Limited** (different performance tiers, premium performance is expensive; manual data sharding possible)	**Unlimited** (requires correct use of partition key)	**Unlimited** (requires manual data sharding to multiple collections)	**Limited** (different performance tiers)
Throughput	**Low** (200 concurrent request for standard S3 tier; max. 2 400 request for premium P11 tier)	**High** (1 000 req/s per partition)	**Medium** (250-2 500 req/s per collection depending on performance tier)	**Very high** (600-250 000 req/s depending on performance tier)
Cost	Storage: **low** Performance: **high**	Storage: **very low** Performance: **very low**	Storage: **medium** Performance: **low**	Storage: **very high** Performance: **medium**

abstraction techniques (O/R mappers etc.), which often leads to unnecessary overloading of the database because developers easily lose track of the associated load costs.

To increase the scalability of a software component or service, Taylor et al. [8] recommend that system bottlenecks should be avoided by design. In case of the cloud application the bottleneck is the relational database with limited throughput. First we should question whether we can easily lower the database load by redesigning the application logic to minimize the number of needless DB calls per user request. Furthermore, we should consider replacement of the relational database with another cloud storage service (e.g. NoSQL database), which may however not be feasible due to the required storage feature set. Relational database is the only storage providing complex transaction support with built-in support for data integrity enforcement (most commonly unique and foreign key constraints), which are both desirable features of the primary data storage.

*2) **Tactic Description**:* This tactic is described as a software design pattern that generates prepopulated views over data in one or more data stores when the data is formatted in a way that does not favor the required query operations. This pattern can help to support efficient querying and data extraction, and improve application performance [6].

A mechanism of a materialized view (indexed view) is also used in relational databases, in a way that a normal database view (stored SELECT statement) inlines its definition to a calling query and the database server evaluates a single query against stored data, giving database view no performance benefit. In case of materialized view, the query in its definition is evaluated and results are persisted in form of a table, which is recomputed with every change of source data. This approach, when properly used, can bring significant performance benefits, especially when applied to views with aggregations over huge data sets or views with complex queries involving joins over many tables, which are less frequently updated. The more frequently the data is read between updates, the higher is the efficiency of this tactic.

In the cloud, we would like to offload the relational database service, therefore we may use this tactic to precompute views on data which will be stored mostly in NoSQL databases due to their high scalability. Application can then load data from a highly scalable storage in a form which requires minimum processing at the application server and can be immediately sent to the client.

*3) **When not to use**:* This tactic should not be used when source data is frequently updated, because each update triggers either complete or partial updates of the materialized view. This may not be an issue when updates are done asynchronously, but when implemented synchronously (due to strict data consistency requirement), the performance of data modifications significantly decreases. The guidance on when to use this tactic is provided in *Table II*.

*4) **Our implementation**:* The cloud application we use for demonstration employs this pattern to combine two storage services—Azure SQL Database and Azure Table Storage. The objective is to combine those services in such a way that their strengths (see *Table I*) prevail and weaknesses are mitigated.

First we designed the relational database based on the data model of our application. During the database design we payed attention to ensure that the designed database tables are in a normalized form to store data efficiently and without redundancy. We did not make any optimizations on the schema level to support more efficient querying of the data except index design.

TABLE II
GUIDANCE ON THE MATERIALIZED VIEW TACTIC

		Storage read frequency	
		low	high
Storage write frequency	low	Small benefits relatively to implementation effort, use a single storage	Most benefits, low issues
	high	No benefits, use single storage (relational DB) and if not sufficient use Indirect Dependency Tactic to load data to storage	If strict data freshness required, then not applicable; otherwise use Indirect Dependency Tactic to load data and refresh cache in an asynchronous manner.

Based on our analysis, most of the load will be caused by the subsystem which is responsible for storage and retrieval of wall records (95% of expected API calls). Therefore we focused on the wall record entity and how it is displayed in the application. When the user loads the wall, the last 15 wall records are displayed in a compact form including preview of the text and preview of the last two comments. We discovered that the displayed data is (in the normalized DB schema) loaded approximately from 10 tables. If such a complex query or set of queries was executed every time the data is requested from the client, the throughput of this service would be limited to less than 100 requests per second (due to the limited throughput of the database). Vertical scaling of the database is not an option because of expensive high-performance tiers in comparison to other storage services.

To offload the traffic to Azure Table Storage, it was necessary to circumvent the limited querying capabilities of this service to form queries retrieving data based only on the partition and row key. The application logic, which decides which posts and in what form are visible to a specific user, is too complex to be evaluated during every call. Therefore we store in the Azure Table Storage for each user a copy of his completely prepopulated wall. This gives us extremely efficient querying on cost of high data redundancy, which does not cause (due to storage cost $0.07 per GB) any significant additional operation cost. Design of the data partitioning strategy is discussed in the *Section IV-B*.

It is important to discuss in what form we store entities in the Azure Table Storage to introduce other related benefits. Our goal was to offload processing from the web servers hosting the API to increase their throughput. We achieved this by storing wall record entities in the most complete form so that it contains all data attributes necessary for construction of any variant of the wall record data transfer object (DTO) which is sent to the client. Two forms of the wall record DTO can be send from the server—a compact form which is displayed as an item on the wall and a detail form which is displayed when the user selects post on the wall and opens its detail. We persist all records in the full-detail form as the limited one can be easily derived from it by limiting the set of attributes.

All complex transformations are computed and persisted when a new record is stored in the Azure Table Storage and because the record is read in order of magnitude more often than it is updated, it significantly saves resource expenses.

To sum up, our design decreases the load on the wall by:

- Submission of a single query to Azure Storage.
- Retrieval of the first 15 records using the most efficient type of query offered by this service (or next 15 records based on the continuation token).
- Copying some data attributes from objects returned by the query to final data transfer objects without further processing.
- Return of results to the client.

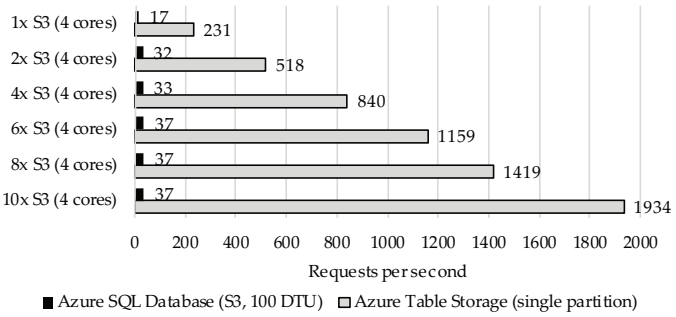


Fig. 2. Comparison of API throughput with and without materialized view

*5) **Tactic impact**:* To evaluate the impact of this tactic on the throughput of the optimized API, we conducted a set of performance test (which loaded the wall of a single user) with and without the use of this tactic, with results presented in *Figure 2*. The API was hosted in Azure App Service using S3 instance size (4 CPU cores per instance).

Positive effect of this tactic is in our case clearly measurable and allows us to scale this service in the desired extent.

Negative effects of this tactic related to significantly increased complexity of data modification had to be mitigated by application of the Command and Query Responsibility Segregation in combination with asynchronous write processing using messaging, as described in later sections.

*6) **Our contribution**:* Our contribution to this architectural tactic is the identification of its surprisingly massive impact on scalability and throughput of the PaaS cloud application, from which many read-intensive applications can benefit. We have also identified that despite the fact that the use of this tactic is recommended together with the Command and Query Responsibility Segregation (CQRS) pattern (see below), it is nowhere recognized that this tactic becomes nearly useless without asynchronous update processing via messaging.

B. Sharding Pattern

*1) **Motivation**:* Applications often store data in a single storage. But the problem is that such a single storage can in most cases efficiently scale only vertically—for instance adding more CPU power to the database server which operates the database. The main issue is that vertical scalability has its limits in term that not much processing power or storage can

be added to a single storage node. When we reach the limit or upgrade costs exceed certain efficiency (as costs of high-end hardware does not rise adequately with the performance) we will have to separate the data into more storages or databases to scale horizontally and overcome the vertical scalability limit.

*2) **Tactic Description**:* In [22] the Sharding pattern is described as a division of data store into a set of multiple horizontal partitions (shards). For each row, its partition is selected based on a value in a partition key, which is one of the columns in the table. Partitions are then distributed and stored across multiple servers, therefore the load may be distributed, until queries work only with a single partition. Partition key should be selected in such a manner so that the load is evenly distributed across as many partitions as possible and at the same time most of the queries are working with data from a single partition.

*3) **When not to use**:* This tactic should not be used when structure of stored data or data access patterns disallow the separation of data set into distinct partitions because of frequent queries that would work with data from multiple partitions. That is because this is for most storage technologies inefficient due to synchronization overhead, when results from multiple nodes have to be collected together.

*4) **Our implementation**:* Because we have used Azure Table Storage as the storage for data generated by the *Indexed View architectural tactic*, it was crucial to design a robust partitioning strategy, as this is in the case of Azure Store the key for unlocking its unlimited scalability potential. Azure Table storage uses two-part shared key with the following properties:

a) Scalability: Rows with the same partition key are stored at the same storage server. Rows with different partition keys may be stored on different servers. This service is responsible for internal data distribution across hundreds or even thousands of servers in the data center. To unleash the unlimited scalability of this service it is necessary to distribute data across reasonable number of partitions depending on the load.

b) Query efficiency: The most efficient queries (with low response time) are when the filter condition contains an exact match of the partition key and at the same time an exact match of the row key (fastest) or range match of the row key (if range is not too wide) or returns only first few records. Queries across multiple partitions have declines in the response time. Performance of queries filtering based on other non-key fields is inconvenient.

c) Batch updates: It is possible to modify multiple rows with the same partition key in the batch gaining significant performance benefit.

d) Results order: Data is returned automatically in an ascending order primarily by the partition key, secondarily by the row key.

e) Comparison operator: Both keys are strings and as strings are also compared (including inequality operators greater than, etc.). This requires us to use numbers with fixed digits count and leading zeros.

For the *partition key* we decided to use internal user identifier which is provided for each API request by the authentication system. This gives us the desired scalability of read operations as data of each user may be stored on different server. When multiple users load data, their load is automatically evenly distributed across high number of servers. At the same time we always load data for a single user and a single partition queries have desired response time.

The *row key* must be in combination with the partition key unique in the table. It an advantage when it also ensures, that wall records for a given user are returned in descending order based on time of their creation. This order is advantageous because the most frequent query is expected to be the load of *n newest* record on the user's wall which can be now translated as load of *n first* records in the partition.

The biggest challenge related to this tactic for us was how to implement data filtering with multiple criteria in the Azure Table Storage which supports only queries filtering based on partition and row key. Because the application supports the concept of class and subject walls, which are intended for communication only with enrolled students, we need to filter wall record accordingly to provide this type of views in addition to the global view, which contains all wall records visible to the user. In both cases we need to filter records based on their type—private message, homework and poll.

To achieve this functionality, we need to store multiple lists of wall records for each user based on filtering criteria. This tactic increases data duplicity (which is not an issue due to very low storage costs) but provides very effective data retrieval. To support filtering based on type of the record, we created in the Azure Table Storage multiple tables, one for each record type plus the global one. The same wall record is stored in multiple tables based on its type (for instance a record of type homework definition is stored in tables WallRecords and WallTasks). Now it is very simple to load all types of records or just a specific one only by changing table name in the query without any performance or scalability degradation.

If the wall record is published for the class or for the subject, additional copy is created and '-class-{ClassID}' or '-subject-{SubjectID}' is added as a suffix to the partition key allowing efficient filtering based on class and subject.

*5) **Tactic impact**:* In our benchmark we were able to achieve a total throughput of a single partition in the Azure Table Storage over **6 000 queries per second** retrieving a single entity, which is a surprising result, because the number of retrieved entities per second is three times higher than the limit specified in the service documentation [23].

We also measured an average response time of Azure Table Storage, which is **29 ms** for lookup of any entity based on partition and row key. When the entity was recently loaded, average response time is **9 ms**.

The primary disadvantage of this tactic is a high redundancy of stored data, where a wall record of type homework defini-

tion published to a class with 30 students and one teacher is stored in the Azure Table Storage in $31 \times 2 \times 2 = 124$ copies (not identical records, they vary in their keys, but each copy takes into account the user, their permissions, role in school, etc.). Because the indexed view needs to be updated with every change in any wall record or after posting a new one and this update was previously implemented synchronously, API calls modifying the wall started to time-out which is unacceptable therefore we had to implement asynchronous updates using messaging.

*6) **Our contribution**:* Our main contribution is in the identification and demonstration of an alternative practice for storing data, which from the initial point of view of many developers experienced with relational-database development seems to be absolutely inefficient due to needless data redundancy. This was the same in our project where after mind-switch of our developers it was surprising for everyone that the NoSQL databases fulfilled all the quality requirements—high scalability and low response time for read operations and low operation costs, despite the redundancy.

C. Static Content Hosting

*1) **Motivation**:* Even though web applications are currently composed mostly of dynamically generated content, some resources of the web application are static resources such as images, style sheets, client side scripts or separate downloads (for instance PDF or video files) [22]. Very common anti-pattern we observe is, that both dynamic and static content is delivered by a single server which is optimized for dynamic content generation.

In the cloud environment application servers suitable for generation of dynamic content are billed based on their performance. If we unnecessarily use application servers capable of dynamic content generation for distribution of the static content, we waste valuable resources thereby also increase operation costs despite the fact that cloud providers offer highly scalable and cost efficient services dedicated for distribution of the static content including support for scalability extension using content delivery network services.

We see increased need for application of this tactic because it can significantly decrease operation costs of many novadays popular single page applications, which consist of static resources including client side scripts that load dynamic content served through REST API which is in many cases the only component of the application requiring compute resources associated with a specific application server.

Another anti-pattern we observed is that static content (most often images) is stored in the relational database and is loaded by by the client through the application server. The motivation for this approach is a development comfort associated with a single unified storage, that is redeemed on cost of expensive non-scalable data storage for binary data and unnecessary additional load of expensive application servers. For applications depending on a single relational database is this anti-pattern very dangerous as it may significantly affect performance of entire service depending on the overloaded database.

*2) **Tactic Description**:* This tactic encourages developer to separate hosting of the static content from the dynamic content of the same web application thus decreasing demand for expensive compute resources [22].

*3) **When not to use**:* Challenges related to this tactic are following:

a) Restricted access: For sensitive static content it is necessary to apply user authentication and authorization when accessing the resources. Use of the same authentication mechanism as for the rest of the application may be applicable on proprietary web servers hosting the static content, but most of the cloud providers do not provider an option to implement custom authentication for their storage services. This issue may be solved with the help of the *Valet Key* tactic.

b) More complex application deployment: As the web application itself is deployed across multiple services, separating dynamic and static content the deployment process becomes more complex and it is desirable to automate application deployment to prevent mistakes.

c) Limitations of the storage service: Deployment of this tactic may become complicated due to lacking support of required technologies at the storage service side—HTTPS support, custom domain support (especially in combination with HTTPS). Missing custom domain support requires proper handling of cross-origin resource sharing at the application level.

*4) **Our implementation**:* Because the web client of our application is a single page application, we take advantage of this tactic to store all publicly available static content in a public container of Azure Blob Storage service. This freed capacity of application servers and increased throughput of our REST API.

*5) **Tactic impact**:* Use of specialized service for distribution of static files instead of application server offers significant throughput advantage as depicted in *Figure 3* and response time advantage as depicted in *Figure 4*. Performance measurements were conducted using *wrk tool* a HTTP benchmarking tool, running on Azure 8 CPU core VM in different data center, using 12 threads and 256 connections for the benchmark. In the test 85 KB JavaScript file was downloaded.

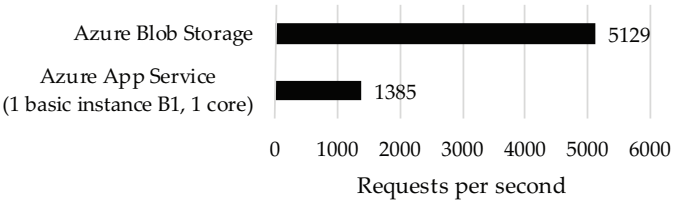


Fig. 3. Throughput of Azure Blob Storage vs. Azure App Service.

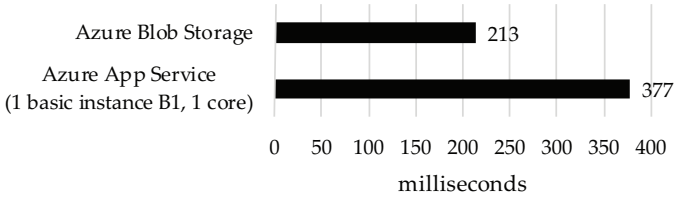


Fig. 4. Response time of Azure Blob Storage vs. Azure App Service for 90% fastest requests.

*6) **Our contribution**:* Our contribution is in the quantification of the performance advantage of the dedicated storage service. Our benchmarks discovered an insurmountable throughput limit of 2 200 requests per second even for overprovisioned services. This unexpected and surprising limit is likely caused by the performance limit of the loadbalancer which is a part of the platform. With the discovery of this limit, the importance of this tactic significantly increases, as many small file transfers served by the same application server as the API can easily saturate the loadbalancer, thus decreasing the throughput of the API and keeping the application servers underutilized and cost inefficient.

D. Command and Query Responsibility Segregation

*1) **Motivation**:* Common practice for current web and line-of-business applications is that the application has a single data model which is used for both data querying and data manipulation (CRUD[1]).

As our needs has become more sophisticated, we have realized that having unified model and data transfer objects (DTO) for read and write operations brings some issues:

- When reading data, we may want to read the information from the data store in a different form—applying aggregations, adding computed attributes and at the other hand hiding some internal attributes, forming virtual records, etc. The same information can have multiple representations.
- When updating data we usually need to apply additional validation rules and we may want to update attributes of the entity which are never again read in their original form therefore missing in the read model where are present for instance only in their recomputed form.

*2) **Tactic Description**:* Command and Query Responsibility Segregation (CQRS) is a pattern that segregates the operations that read data (Queries) from the operations that update data (Commands) by using separate interfaces. This implies that the data models used for querying and updates are different [22].

*3) **When not to use**:* In case of this pattern, it is important to understand, that using a single data model for read and write operations is not automatically a bad practice just because there is a pattern that can separate them. Implementation of CRUD using a single data model is a well suitable practice for many applications which does not bring artificial complexity to their software architecture.

CQRS pattern should only be used when the application will get performance benefits by its implementation or the read and write model are significantly different. Otherwise this pattern will significantly increase software architecture complexity and related development costs.

*4) **Our implementation**:* When we implemented the first version of the materialized view tactic in our application, we realized that it is necessary to capture all data modifications which lead to an update of the materialized view in a highly structured way not to get lost in dependencies. We found out, that capturing data modifications at the repository level is not a good practice because many update operations are complex transactions which consist of multiple individual operations and we needed more precise control over updates of the materialized view.

Therefore we created a separate command class for each update operation that can be submitted via the API by the user of the service. Other important decision is, that we separated the command into 2 classes—one class contains only serializable inputs of the command, the second is only a command implementation that receives an instance of the class with inputs as a parameter of the Execute method.

By implementation of isolated update commands we naturally separated read and write model for performance critical operations. The read model is reading most of the data from indexed views in Azure Table Storage. Few operations where a high rate of request is not expected still query directly the relational database (if operation will prove problematic with increasing load, database access will be replaced by the materialized view).

It is important to mention, that not all components of the application use CQRS—the administration interface with very low expected load is written using a single data model for CRUD operation. Only in case any operation would impact validity of any materialized view, specific command to update the materialized view is executed.

Despite the implementation of CQRS which in our case made the software architecture more transparent we still encountered significant performance issues (their roots explained in Section IV-B5) when the update commands were executed synchronously by the application server which processed the API calls. Because the CQRS pattern itself does not specify how the command should be executed we decided to shift from synchronous processing to message based asynchronous processing.

*5) **Tactic impact**:* The CQRS tactic itself does not have a direct impact on a performance of the application as it is just the way how we restructured the code without any change to its functionality. Performance issues of update operation are measured in Section V-A5. But application of this tactic is a required condition to shift from synchronous to asynchronous command execution which is intended to significantly improve update performance.

[1]Create, read, update and delete operations

*6) **Our contribution**:* Our contribution is in a seamless combination of this patterns with other described tactics which leads to significant performance improvements and well-organized software architecture with clear dependencies.

V. Messaging and Data Processing Tactics

In this section we present tactics related to a design of asynchronous messaging infrastructure we have used in the implemented project. Our primary motivation for implementation of asynchronous messaging patterns in the project were performance issues bound to updates of complex materialized views, which caused synchronous update commands sent by the client even to time-out. Because of poor scalability and throughput of update operations we had to rethink the architecture in a way, that would lead to a reliable high throughput read and write operations.

A. Asynchronous Messaging

*1) **Motivation**:* Software architecture of an application determines how an incoming request from the client is going to be processed, which components are called in which order, how the results are being generated and transferred back to the client. Most of the current web applications utilize synchronous request processing, i.e. when a request is received by the web server, it is put into a queue of incoming requests in the web server and is processed as soon as the web server has an available processing thread. Request processing is done synchronously, the request is blocking the processing thread until the result is generated and the response is sent back to the client. This involves synchronous calls to the database storage, and execution of methods generating the output.

Synchronous request processing has an advantage in low development costs, because it does not introduce additional complexity into the architecture. The main disadvantage is low scalability because every incoming request that is immediately processed has many associated resource costs and when certain resource becomes overloaded, processing of the request fails due to time-out, often with no retry mechanism being implemented. In case there is a load balancer in front of the web servers, random load distribution among web servers may in case of resource intensive requests lead to uneven load distribution, with some servers being overloaded and the remaining being idle.

*2) **Tactic Description**:* An opposite approach, described by the Indirect Dependency Tactic [8] or Asynchronous Messaging Primer [22] is to receive a request, store this request in a queue service and let a continuously running worker withdraw requests from the queue and process them. This architectural tactic is recommended as a way to increase system scalability by utilization of indirect dependencies.

*3) **When not to use**:* For a software architect it is challenging to assess if the benefits of asynchronous request processing in the designed application outweigh the drawbacks in terms of the increased architecture complexity and higher development and operation costs.

Application of this tactic becomes unfeasible when the client requires immediate response which is based on results of asynchronously executed command, for instance return of inserted record identifier which is generated in the database as a part of transaction which stores the record.

*4) **Our implementation**:* To solve our problem with the performance of update commands we altered the way, how these commands are processed by the application server and when they are executed. To increase performance, the application server does only very basic command validation which includes:

- Authentication of the user
- Authorization for the operation but only in the case that it does not require complex permissions evaluation otherwise is authorization part of message processing
- Basic validation of received command which does not require data access
- Storing of the serialized command in the scalable queue service

To process queued commands we implemented and deployed a worker role, which is a stateless virtual server where our application is deployed, which in an infinite loop fetches commands from the queue and executes them.

During implementation of this tactic we identified multiple trade-offs we had to deal with. The major challenges related to this tactic includes:

a) New record identifiers: When a user inserts a new wall record, the record appears on the wall as soon as the API confirms that it has successfully received the request. It passed basic validations and the command is stored in the queue, so that it is guaranteed that it will be processed by the worker. The problem is that primary key of the record is generated in the database (due to benefits associated with numeric auto increment primary keys), and even write to the database is done asynchronously, therefore this operation cannot return identifier of the inserted entity to the client. But when the record is visible to the user, user can invoke another operations (for instance mark the post as favorite) which require a record identifier. To solve this issue we decided to add an GUID alternate key to the entity which is used only by the client instead of the primary key. Benefit o this alternate key is that the API is responsible for its generation, thus it is stored in the insert command and immediately returned to the client and the database only enforces its uniqueness.

b) Results from asynchronous operations: Because most of the resource intensive operations are processed asynchronously the client is not aware of their completion and cannot consume immediately their results. On the other hand results visible to the client are written by asynchronous operations to the database and mostly to indexed views but still the client does not receive the impulse to download them.It is inefficient to poll the API for new data every few seconds therefore we used web sockets technology to allow the worker to send notifications directly to the clients and efficiently notify them about completed operations. This approach has brought a

completely new level of interactivity and efficiency, because as a part of the change notification we deliver via the web socket straight the modified entity which is immediately displayed by the client. When somebody inserts a new wall record, based on the permissions this record is automatically displayed to other connected clients with a few seconds delay.

c) Interoperability: It was an implementation challenge to connect REST API of our service which is implemented in the PHP with worker process implemented in .NET. We had to define identical classes holding command input data as explained in *Section IV-D4* both in PHP and .NET and ensure, that instance of the class is serialized to JSON and stored to the queue in the same way as it is loaded in the .NET worker process where it is deserialized from JSON into instance of the same class and passed to the command to execute.

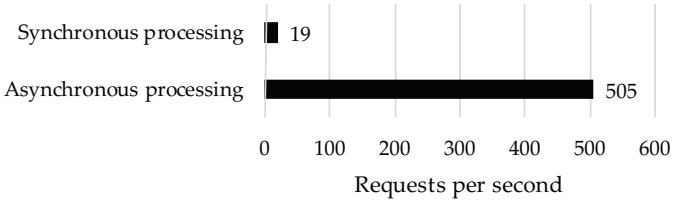


Fig. 5. Comparison of the API throughput between synchronous and asynchronous inserts of new wall records.

*5) **Tactic impact**:* As depicted in Figure 5 this tactic had a significant impact on performance of update commands which lead to update of indexed views.

*6) **Our contribution**:* Our contribution related to this tactic is in the implementation of infrastructure stack for asynchronous messaging which significantly simplifies implementation of this tactic and individual commands. Another contribution is a solution of the described problems related to missing record identifiers and efficient use of real-time notifications via web socket technology which have not been discussed in other sources at sufficient level of detail yet.

B. Competing Consumers

*1) **Motivation**:* Implementation of asynchronous messaging tactic will lead to uneven allocation of processing power in case of multiple messages produces and a single message consumer, which may become a bottleneck in the architecture, significantly overloaded in time periods of higher load as the messages pile up in the queue. Despite the fact, that asynchronous messaging tactic will guarantee that none of the client requests will timeout, as the responses are generated immediately, the message stored in the queue may be processed after significant delay which may not be in compliance with user requirements and specified quality criteria.

*2) **Tactic Description**:* Competing consumers tactic is a natural extension of *asynchronous messaging tactic*, which enables multiple concurrent consumers to process messages received on the same messaging channel. This tactic enables a system to process multiple messages concurrently to optimize throughput, to improve scalability, and to balance the workload [22].

*3) **When not to use**:* This tactic is not suitable when the messages (tasks, commands) have to be processed in an exact order. Because of parallel processing it happens that due to different processing times of individual messages, tens or hundreds of messages can outrun a single long processing message.

*4) **Our implementation**:* Implementation of this additional tactic which is an extension of asynchronous messaging tactic was very straightforward, because the worker roles service we used for hosting of the worker process supports horizontal scalability, which allows deployment of the identical application code on multiple virtual servers. Because the instances of the worker process share also configuration they all point to the same queue from which they load messages for processing.

Despite great increase of scalability, during implementation of this tactic we came across multiple architecture and algorithm design problems that we had to resolve:

a) Message ordering: When multiple workers process messages before a single message is completed multiple newer messages can be already processed, which is an issue when messages have dependencies and should run in a guaranteed order. We have solved this issue by letting a message during its processing submit other messages to the queue, which gives us guarantee of the order in which the messages are processed in case it matters.

b) Idempotency: Our implementation of the queue service provides a guarantee that message will be processed at least once, but not exactly once. Despite the fact that a message is locked in the queue when a worker loads it, to prevent other workers from processing the same message, it may happen that due to an error in processing, the message is rescheduled and processed once again. Therefore the system should be designed in a way that this causes no problems—for instance insertion of a duplicate record.

c) Poison messages: A poison message is a message that cannot be successfully processed by the worker and usually causes an exception during its processing. The .NET client library which we use for loading of the messages from the Azure Service Bus Queue service automatically detects when a message is loaded form the queue and its processing fails multiple times. Such a message is automatically moved to the Dead Letter Queue where it waits for further analysis by the developer of the service. We consider it important to analyze messages in this queue as they can reveal bugs in the system.

d) Throughput of the messaging service: The messaging service has its throughput limits we have to be aware of. We recommend to use batch processing when messages are written to the queue. It is possible either to increase a queue throughput by separating the load into multiple queues or decrease the load by reading messages in batches.

e) Message scheduling: Azure Service Bus Queue supports submission of messages that becomes visible to workers

at specified time. We use this feature to implement some of the future actions—for instance deadlines to submit homeworks. Every time somebody sets a deadline for homework we schedule a message with a command that becomes visible few seconds after the deadline. This command on the beginning of its execution has to check its validity (if the deadline was not changed again) and refresh indexed views so that users see the homework locked.

*5) **Tactic impact**:* We consider this tactic to be an advantageous extension of the asynchronous messaging tactic which significantly improves its scalability. Especially in the cloud environment, we can benefit from elasticity of the cloud and easily horizontally scale the number of workers consuming and processing messages from the queue.

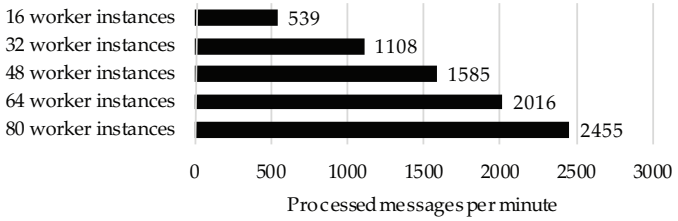


Fig. 6. Throughput comparison of worker processes depending on number of deployed instances. This measurement illustrates

Scalability of this tactic is demonstrated in Figure 6, where the numbers of processed messages per minute are compared based on the number of deployed worker servers. It is evident that the throughput of our worker layer scales nearly linearly with the number of processing nodes.

*6) **Our contribution**:* Our contribution to this tactic is in a solution of described issues, especially how we deal with a message ordering and idempotency, which allows us to deploy workers at the massive scale as demonstrated without risk of data corruption by parallel processing of queued commands. Our approach to message ordering also allows us to divide complex commands into a set of small atomic commands, which can be in case of failure repeated which leads to an increase of resiliency of the asynchronous message processing. Another contribution is in our approach to idempotency, when we clearly divided commands into 2 groups: the first where are commands which cannot be repeated due to the nature of the operation that would create duplicates records—those commands are protected by using unique identifiers generated at the time of command scheduling which blocks addition of the same record twice, and the second group where are commands which can be safely repeated.

VI. Conclusion

In this paper, we have reported on our experience with the architecture design of PaaS cloud applications, and destiled our observations into a set of architectural tactics (based on existing tactics used in other domains), which are aimed at supporting software architects in their PaaS cloud application design. As distinct to related work, we have focused on both pattern and anti-pattern effects of the studied tactics, and their combinations, highlighting their impact on the quality characteristics of the cloud applications based on PaaS infrastructure.

In future, we would like to implement an automated support for the evaluation of the tactics and their impact on various quality characteristics of cloud applications in the PaaS context.

References

[1] P. Mell and T. Grance, "The nist definition of cloud computing," 2011.

[2] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[3] M. Fowler, *Patterns of Enterprise Application Architecture*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[4] T. Erl, R. Puttini, and Z. Mahmood, *Cloud Computing: Concepts, Technology & Architecture*, 1st ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2013.

[5] B. Wilder, *Cloud Architecture Patterns*, 1st ed. O'Reilly Media, 2012.

[6] A. Homer, J. Sharp, L. Brader, M. Narumoto, and T. Swanson, *Cloud Design Patterns: Prescriptive Architecture Guidance for Cloud Applications*. Microsoft patterns & practices, 2014.

[7] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. Addison-Wesley, 2003.

[8] R. N. Taylor, N. Medvidovic, and E. M. Dashofy, *Software architecture: foundations, theory, and practice*. Wiley Publishing, 2009.

[9] J. Sahoo, S. Mohapatra, and R. Lath, "Virtualization: A survey on concepts, taxonomy and associated security issues," in *Computer and Network Technology (ICCNT), 2010 Second International Conference on*, April 2010, pp. 222–226.

[10] J. T. Piao and J. Yan, "A network-aware virtual machine placement and migration approach in cloud computing," in *Grid and Cooperative Computing (GCC), 2010 9th International Conference on*, Nov 2010, pp. 87–92.

[11] N. M. K. Chowdhury and R. Boutaba, "A survey of network virtualization," *Computer Networks*, vol. 54, no. 5, pp. 862 – 876, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128609003387

[12] R. Buyya, R. Ranjan, and R. Calheiros, "Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities," in *High Performance Computing Simulation, 2009. HPCS '09. International Conference on*, June 2009, pp. 1–11.

[13] T. Ganesh. (2012, Jun.) Designing a scalable architecture for the cloud @ONLINE. [Online]. Available: https://www.ibm.com/developerworks/community/blogs/theTechTrek/entry/designing_a_scalable_architecture_for_the_cloud3?lang=en

[14] J. Varia, "Architecting for the cloud: Best practices," *Amazon Web Services*, 2010.

[15] B. Jimerson, "Software architecture for high availability in the cloud," *Oracle Technology Network*, Jun. 2012.

[16] Y. Nelapati and M. Weiner, "Scaling pinterest," InfoQ, 2013.

[17] S. Anand, "Netflix's cloud data architecture," InfoQ, 2011.

[18] A. Sobel, "Scaling the social graph: Infrastructure at facebook," InfoQ, 2011.

[19] M. Cecconi, "The architecture of stack overflow," code.talks, 2011.

[20] "Wikimedia servers," =https://meta.wikimedia.org/wiki/Wikimedia_servers, 2015, accessed: 2015-10-25.

[21] "Schools and education institutions in school year 2014/2015 - table 7: Elementary schools - number of students by region," =https://www.czso.cz/documents/10180/20555525/2300421507.pdf/a0e9a81f-e13a-4423-b260-410db2da70fe?version=1.0, 2014, accessed: 2016-01-13.

[22] S. Lehrig, H. Eikerling, and S. Becker, "Scalability, elasticity, and efficiency in cloud computing: A systematic literature review of definitions and metrics," in *Proceedings of the 11th Int. ACM SIGSOFT Conference on Quality of Software Architectures*. ACM, 2015, pp. 83–92.

[23] "Azure storage scalability and performance targets," =https://azure.microsoft.com/en-us/documentation/articles/storage-scalability-targets/, 2015, accessed: 2016-01-23.