

Lappeenranta University of Technology
Faculty of Technology Management
Degree Program in Information Technology

Master's Thesis

Janne Parkkila

**OPTIMIZING CUSTOMER ENERGY INVOICE CALCULATION
WITH PARALLEL COMPUTATION**

Examiners : Professor Jari Porras
Adjunct Prof. Jouni Ikonen

Supervisors: Professor Jari Porras

ABSTRACT

Lappeenranta University of Technology
Faculty of Technology Management
Degree Program in Information Technology

Janne Parkkila

OPTIMIZING CUSTOMER ENERGY INVOICE CALCULATION WITH
PARALLEL COMPUTATION

Master's Thesis

2012

66 pages, 15 figures, 19 tables

Examiners: Professor Jari Porras
Adjunct Prof. Jouni Ikonen

Keywords: parallel computing, distributed computing, cloud computing, invoice computing

The thesis takes a look at optimizing customer energy invoice calculation performance with the use of distributed computing. As the smart energy meters are installed to every household, the energy companies need to calculate customer invoices according to hourly received measurement data. In order to calculate the customer invoices correctly within a small amount of time a metering tree algorithm was devised. In addition, the thesis evaluates distributed computing options for optimizing the computation and takes a closer look at cloud computing. An amount of simulations were done to evaluate the advantages of parallel computation against the traditional sequential computing.

PREFACE

I wish to thank my supervisors Jari Porras and Jouni Ikonen for their support and guidance on writing this thesis and during all my studies. Thanks for giving me the inspiration of programming on those codecamp courses! And thanks for enduring with my problems on finishing this thesis!

Also, I want to thank my classmates with whom we have sat all these years; Johannes, Anssi, Ville & Rosti. Those were the days.

Thank you Rostislav for attending all the tough courses with me. Without your support and friendship I would never have found my passion for programming and got this far.

A special thanks goes to Pedro Larrañaga and Concha Bielza from the Universidad Polytechnica de Madrid, who taught me important lessons of becoming a researcher.

Thanks to my brother Aleksii for all the long discussions we have had on studies and for all these years. Thank you for sharing the hard times. православное братство!

Thanks to my parents for supporting me through all these long years. Thanks for the hundreds of meals and clean clothes I've received.

Finally, I want to thank my loving wife-to-be Mikaela. Thank you for staying by my side all these years.

TABLE OF CONTENTS

1	INTRODUCTION	8
1.1	Research Methods and Restrictions	9
1.2	Structure of the thesis	10
2	OPTIONS TO IMPROVE PERFORMANCE	11
2.1	Introducing Different Computing Environments	12
2.2	Computing Environments	13
2.3	Multi-core Computing	15
2.4	Cluster Computing	16
2.5	Grid Computing	17
2.6	Cloud Computing	18
2.6.1	Layers of Cloud Computing	21
2.6.2	What do I need to consider when using cloud computing?	24
2.7	Choosing the right approach	33
3	SYSTEM DESIGN	34
3.1	Smart Metering	35
3.2	System Composition	36
3.3	Proposition for Computation Scenarios	39
3.3.1	Single Invoice Computation Locally	40
3.3.2	Batch Invoice Computation in the Cloud	41
3.3.3	Evaluating Cloud Computing in invoice calculation scenarios	42
4	SIMULATIONS AND RESULTS	47
4.1	Metering Tree Algorithm	48
4.2	Example of Metering Tree Algorithm in Use	50
4.3	Test Environment	51
4.4	Runtime Tests	52

	6
4.5 Parallel Simulations	55
5 CONCLUSION	60
6 REFERENCES	62

LIST OF SYMBOLS AND ABBREVIATIONS

AWS	Amazon Web Services
CPU	Central Processing Unit
EC2	Amazon Elastic Cloud
Gb	Gigabyte
GFLOPS	Giga floating operations per second
Ghz	Gigahertz
I/O	Input/Output
IaaS	Infrastructure as a Service
Kb	Kilobyte
Mb	Megabyte
PaaS	Platform as a Service
RAM	Random Access Memory
RRS	Reduced Redundancy Storage
S3	Amazon Simple Storage Service
SaaS	Software as a Service
SLA	Service Level Agreement
SQL	Structured Query Language

1 INTRODUCTION

In 2009, the Finnish government published an act on providing and measuring electricity for the companies providing electricity [1]. The act defines the responsibilities and requirements of billing and providing electricity. According to the act, 80% of Finnish households should have remotely-readable electricity meters in their households by end of 2013.

At the same time, a research program on revolutionizing the energy markets is underway. The program called SGEM [2], Smart Grids and Energy Markets is aiming to change the energy markets and set new global standards for electricity. The program is carried out in Finland in order to demonstrate the possibilities in a real environment, utilizing the Finnish R&D infrastructure.

One goal of the SGEM program is to bring better service to the clients and allow them to monitor their energy consumption in near-real-time. In addition, with the installations of smart energy meters to customer households, the energy consumption can be measured on an hourly basis. When this is combined with the possibility of charging the clients with the real energy market prices for each individual hour, optimization is required to handle the task efficiently.

As a part of the SGEM program is to create a solution for customers to follow their energy consumptions in real-time and also inform them of the current electricity bill. As the customer energy consumption can be measured on an hourly basis, the price can fluctuate from hour to hour. The energy metering values are stored for each user for each hour. This quickly sums up to a notable amount of measuring data. When combining the amount of data with the requirement of fast response time for calculating user invoices, optimization is needed to handle the task efficiently.

The research problem of this thesis is the calculation of energy invoices for a million customers within two days. Such computation are not unknown to the industry, but the sudden change from old pricing model to near-real-time pricing forces the energy companies to adjust their processes to the new requirements. This thesis evaluates the use of cloud services and the possibility of using them for the invoice computation scenarios. In addition, the possibility of preventing redundant computation in order to improve the processing speed is examined.

1.1 Research Methods and Restrictions

The research carried out is mainly experimental [3] quantitative research with focus on implementing, testing and evaluating different approaches for the optimization problem. The complete research process follows a hypothetical-deductive model [3] where theories of the real-world are used to deduct new hypothesis and solutions. The newly formed solutions can then be applied back to real-world, where the true effect can be verified.

Both local and cloud computation are evaluated as an option for implementing the invoice calculation system. The effects of parallel computation on the processing are investigated by the created simulations. An algorithm for storing and retrieving invoice information is presented as well.

The implementation is done using Microsoft technologies, thus emphasizing on use of Microsoft Azure cloud and .NET framework. The research does not take into account the designing of the database and the effects it has on the performance. Another important factor that is left out from this study is the information security. When dealing with customer information and monetary related issues, the security should never be considered something that can be glued on afterwards. However, as the goal of the research is to find efficient solutions for enhancing system performance and processing speed, the security issue is not covered here.

1.2 Structure of the thesis

The second chapter takes a look at existing options for improving system performance with distributed computing. It shortly covers the possibilities of multi-core, cluster, grid and cloud computing. A more thorough examination of cloud computation is done at the end of the chapter, in order to evaluate its advantages and disadvantages. The chapter also takes a look at two major cloud service providers, Microsoft and Amazon. The two cloud providers are compared in order to gain a better understanding of the available cloud services. This information can be used as a basis for making decisions from the fiscal point of view.

The third chapter takes a look at the design of the invoice computation system. It explains the smart metering system and what kind of data it produces for the purpose of calculating customer invoices. The chapter evaluates two different architectural design approaches to be used in the invoice computation system. Both the usage of local computing and cloud computing are also evaluated for the invoice calculation purposes. The feasibility of these computing solutions are evaluated according to theoretical data of both the size of data transfer amounts and computational complexity of the invoice creation.

The fourth chapter explains the created simulation scenarios and discusses the received results. The devised metering tree algorithm for optimizing customer invoice storing and computation is presented and its usage is tested in the scenarios. The optimization tests the performance advantages of parallel multi-core computing over the traditional single-core, sequential computing.

The final chapter draws the conclusions of the research. It presents the main findings of the research and gives suggestions on creating customer energy invoice calculation system, based on the results received in this thesis.

2 OPTIONS TO IMPROVE PERFORMANCE

Improving computational performance has always been a quest for the Holy Grail. A system could always compute faster and respond to user interaction faster. There is always a small portion left for optimizing and tweaking which means that at some point the system in question just has to be good enough. This chapter takes a look on the available approaches for improving the performance of a computationally heavy system.

One solution to reducing computing times is purchasing newer and faster components [4]. However, that it is not a viable solution alone. Moore's law [5] has shown that the processor speeds do evolve according to time, but a good design can boost performance of the existing hardware a significant amount. Good design in the context of this thesis means both good design of used algorithms as well as good design of architecture. However, the main focus here is on different architectural methods for solving complex computational problems.

An example of good algorithmic design is, the computation of travelling salesman problem [6] is almost impossible to compute with a brute force method even with the powerful computers we currently have. Only with good algorithmic design, such as using ant-colonies [7] can a good enough solution be found. Another simpler example is the use of sorting algorithm. These basic algorithms end up to the same correct result, but each has different computational times. Comparing bubble sort [8] to fast sort [9] shows that fast sort is hundred times faster than the bubble one. This algorithmically different performance is taken into account in the later part of this study.

2.1 Introducing Different Computing Environments

Applications that need to perform complex or otherwise computationally heavy calculations often require more sophisticated methods than the common sequential computing. In the traditional sequential computing, a task is performed on a single, local machine, which handles all the parts of the task in question in a sequential order. That is why it is called *sequential computing*. However, there are other approaches to performing the complex computational tasks than just the traditional one. The complex task can often be split into smaller parts and given out to other computing instances, which then return the answer to the location of origin. This model of computation is often referred as *distributed computing*.

Distributed computing is a term that groups together multiple different approaches that have evolved over time. The common part of all the approaches is that it connects multiple processors together and coordinates the computational efforts [10]. The processors can reside in the same machine, which can have a processor with multiple cores [11] [12] that each performs a portion of the complete task. Another possibility is that the processors are connected together over a network. This means sharing small tasks between multiple computers that all calculate the answers locally and once done, return the answers to the main computer.

Breshears [13] describes the difference of parallel and concurrent (distributed and sequential) computing to be in the processor level. A concurrent system can support two or more actions *in progress* at the same time, while a parallel system can support two or more *executing actions* at the same time. In practice this means, that both systems can support multiple tasks at the same time, but only the parallel one can actually process them at the same time. A concurrent system can alternate between the tasks, but in reality only one is being processed at a time. The most commonly used method for handling concurrent computation is through the usage of threads [14], but these are not covered in depth within the scope of this thesis.

The concept of sharing distributed resources is not a new one. It was already suggested by McCarthy in the 1960s by the statement that "Computation may someday be organized as a public utility" [15]. In his vision the computing facilities operate as a utility, "like a power company or a water company" [16]. The concept [17] [18] of utility computing is interesting and it is funny to notice that the power companies are now looking at information technology to modify their existing models towards a more autonomous and distributed model, as is happening within the SGEM project of which this thesis is also a part of.

Since the 1960s, the data transfer capabilities of networks have evolved a notable amount, changing the discussion of network speed from bauds per second to megabytes per second (or even gigabytes in some cases). So has also the processing power of computers as well as the size of storage space. The newest addition to distributed computing paradigm is the introduction of cloud computing [19] which is one step closer to making computing a utility.

2.2 Computing Environments

The commonly used and known computing environments used in distributed computing are presented here. These are 1) *multi-core computing*, which means computing the tasks on a single machine which has multiple processors available. 2) *Cluster computing* which means computing the tasks on multiple computers that are connected to each other over a local network. 3) *Grid computing* which means sharing of geographically distributed heterogeneous computing resources over a fast network and 4) *cloud computing* which means accessing the computing instances from a service that resides over the internet. Especially the term cloud computing is somewhat ambiguous and a more thorough look is given to it at the end of the chapter.

The different computing environments can be compared according to multiple different scales. Figure 1 shows the difference of the computing paradigms on a scale of

computation scale – service/application orientation. Cloud computing can be seen as solely service oriented approach that can be used in both smaller and larger computation tasks. Clusters are heavily oriented towards smaller-scale applications where supercomputers are seen as highly scalable application oriented approach. Grids fall somewhere in between all of the previously mentioned and do provide an infrastructure that spans across multiple virtual organizations [20]. Grids are often used more in scientific computing, whilst clouds are more common in commercial use. All of these have their best and worst sides.

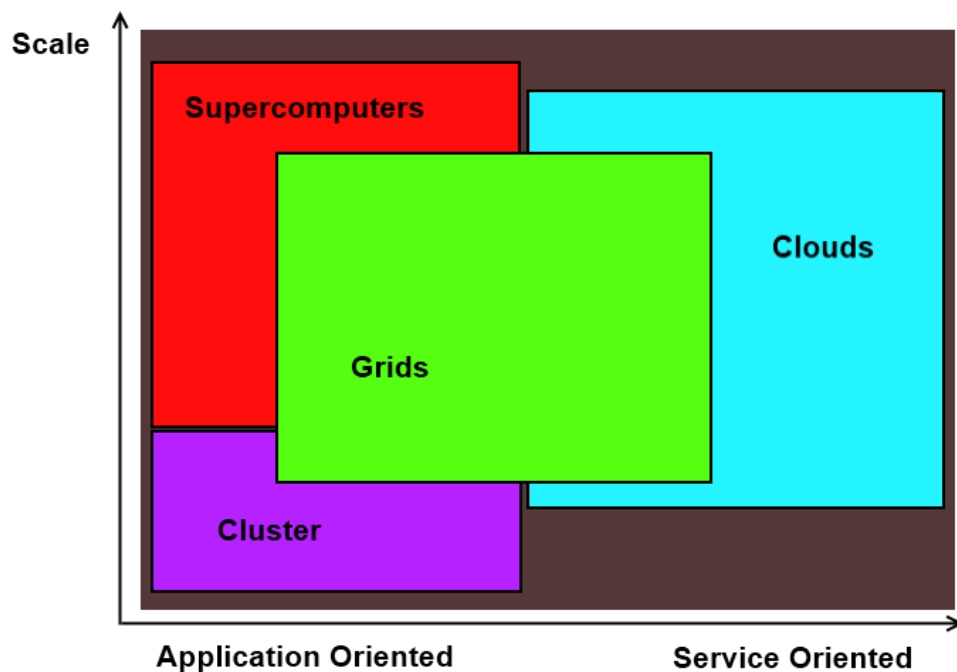


Figure 1 Overview of distributed systems

2.3 Multi-core Computing

Since the beginning of computers, the speed of processors has grown according to Moore's law in a surprisingly fast pace. During the past few decades, the processor manufacturers have had to create new innovations to keep the pace in the ever growing evolution. This has led to adding multiple processor cores on a single processor chip. Nowadays, almost every computer bought from the store is already equipped with a dual-core or multi-core processor. Lately, Intel has been rumored [21] to develop chipset that would have 12 processor cores.

The model of having multiple processor cores on a single processor chip has become the standard of manufacturing. This model delivers better computational performance, lower energy consumption and new capabilities to desktop, mobile and server platforms [22]. The performance of a multi-core architecture is not only limited to computation speed, but answers also to requirements [23] [24] of size, longer battery life and cooling of devices.

The most computation intensive solution of multi-core computing is a super computer. These are often complex systems that are built and optimized for a single-application use [25]. The architecture of a super computer is made often from a set of computers. This usually means having multiple computers with all of the normal components, not just only having hundreds of processors in one machine.

The problem of super computers is often money. For example, India has announced [26] that they are building the world's fastest supercomputer by 2017. The estimated price for the computer is more than 2 billion US dollars. The computer is designed to have performance of 132.8 exaflops (10^{18} floating operations per second). Even though the computation power is enormous, the price alone explains the reason why such computers are not widely used within smaller companies, but instead in big corporations and space associations.

2.4 Cluster Computing

Cluster computing is somewhat similar to multi-core computing. The main difference is, that instead of having only a single computer with multiple processors, the cluster is made of multiple computers that are linked together by a fast local area network [27]. Each of the machines included in the cluster have their own hardware and operating systems but are used to handle computational tasks in the same manner as the multi-core computing.

Clusters of computers offer high performance, scalability, high throughput and high availability at relatively low costs [28]. The clusters can be created from off-the-shelf computers that are available from any stores. This provides a higher availability of computing resources than the regular multi-core computing. Clusters are *local* computing instance, often used to handle computation within a single organization, making them available to all personnel within the premises.

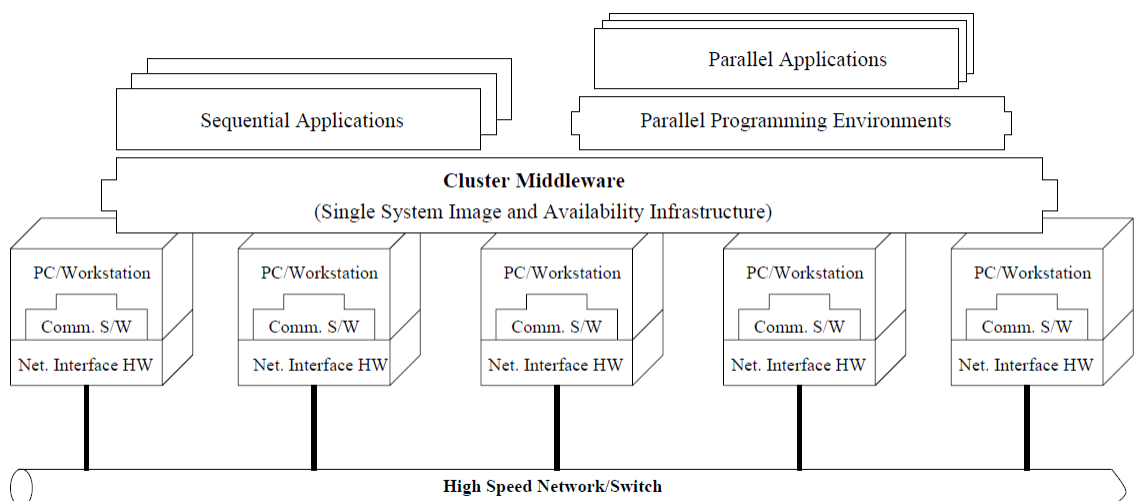


Figure 2 Architectural example of a cluster computer

The main components of a cluster include multiple stand-alone computers, operating systems, high speed network connection, communication software, middleware and applications. An example of typical cluster architecture is shown in Figure 2. The main part of the cluster architecture is the middleware which orchestrates the usage of the computation resources. The middleware is able to run both sequential and parallel applications. The advantage of the middleware abstraction is that the computers used for performing the operations can differ from each other.

2.5 Grid Computing

Grid computing can be seen as a geographically more distributed form of cluster computing. The nodes (computing instances) can reside anywhere in the world, are often loosely coupled and can differ in their system architecture quite much. The connection used between these computers can be either private or public local network or even internet.

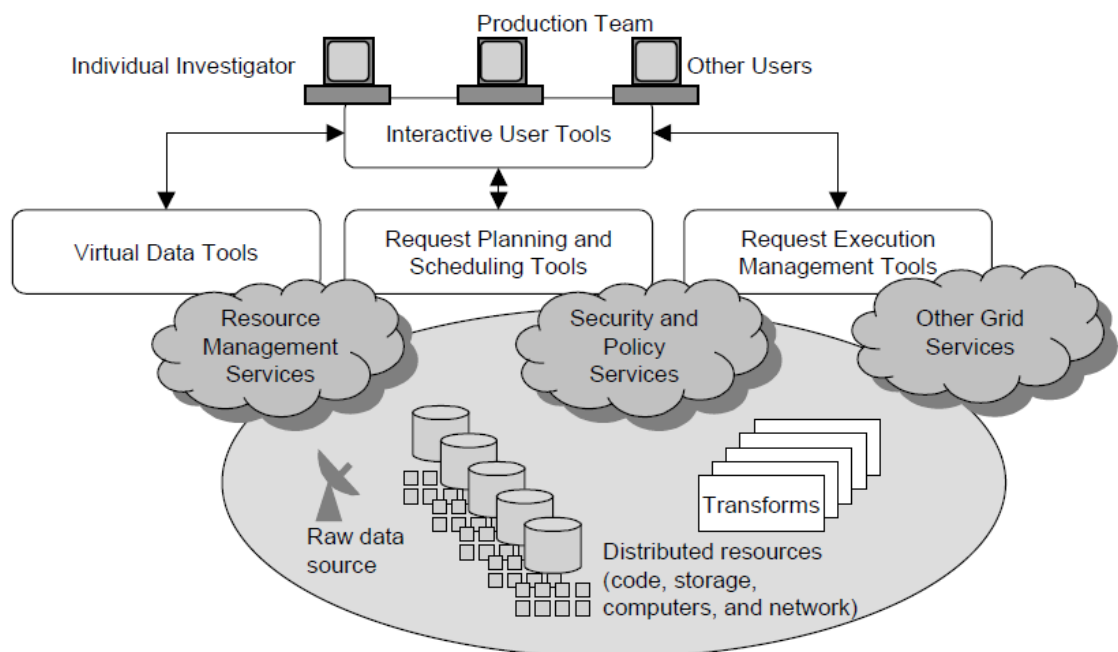


Figure 3 – Example of a Grid architecture

The aim of Grid computing is to offer infrastructure and layers on today's Internet and Web to enable large-scale sharing of resources within distributed, often loosely coordinated groups [29]. The main goal of Grid technologies is to make possible scientific collaborations to share resources of scale and to allow geographically distributed groups to work together in ways that before have been impossible [30] [31]. In other words, Grid is a virtual platform for computation and data management in the same way as the Internet is for information [32]. Grid computing is used in a wide range of applications that require large amounts of processing power, such as Protein Data Bank [33] and Biomedical Information Research Network [34]. An example of Grid architecture is show in Figure 3.

2.6 Cloud Computing

Cloud computing is currently a trendy, but rather ambiguous term which is used for many different systems. The definition used within this thesis is that a cloud is a cluster of distributed computers that provide on-demand resources and services over a network, usually the Internet, with the scale and reliability of a data center [35] [19]. The definition states that cloud computing is a subset of cluster computing. However, it is one step closer towards utility computing [15] than any of the other distributed solutions, because the resources are more openly available over the internet.

The emerging of cloud computing is a combination of multiple advances in the field of information technology. The availability of fast internet connections and the lowered costs of computer hardware have made cloud computing a possibility. As the companies such as Amazon noticed during the mid-2000s that their large computing clusters were idle outside the peak hours [36], they started looking for new uses for the unused resources. The lead to the thought of sharing the idle resources and eventually evolved into the paradigm of cloud computing.

Cloud computing itself is not a revolution. It is a combination of existing technologies that work well together. There is no single clear definition for what qualifies as cloud computing, but the often mentioned [37] [36] main features are *elasticity*, *multi-tenancy*, *economics* and *abstraction*.

Elasticity means the possibility to scale the capacity of the service up or down according to requirements. A company can rent one computer for 100 hours or 100 computers for just one hour to do the same task. The computing resources can also be scaled up for peak hours and then scaled down for quieter times. The possibility of scaling the service is often the most important feature [38] of the clouds that many companies rely on.

Clouds are *Multi-tenant* by their nature, allowing multiple users or workloads to be deployed on the same shared servers. This feature allows taking the full advantage of the cloud, making sure that all the computing resources are in use. There is less idle time and more efficient use of resources.

The *economics* of the cloud means charging the customers according to the time of used computing resources. This allows customers to use only the required amount of resources, lowering the barrier for high performance computing. The Pricing model of the clouds is also related to the on-demand scaling. If a company needs 100 additional computing instances for an hour, it has to only pay for those resources for the hour they were used. This frees the company from purchasing expensive equipment for short-term use and allows organizations to answer swiftly to sudden demand.

Abstraction of the cloud refers to the amount of access to lower levels of the service. The cloud service providers, such as Amazon and Microsoft often provide different service layers to different customers. For example, Software as a Service (SaaS) customer only interacts with the application layer itself, being freed of all the operating system and hardware of the cloud.

All the aforementioned features free the organizations from the previous limitations that have existed. It is no longer economically impossible for small companies to invest to wide-scale server resources, as the cloud services allow for scaling of resources [17] . The organization pays only for the used resources and does not need to worry about the costs of server computers running idle in large server halls. In addition, the developers are not required to have a deep knowledge of system administration as the abstraction allows concentrating on the more important issues.

Cloud computation can also be divided into two different types [35] - the one that provides computing *instances* on demand and the one that provides *capacity* on demand. Both of these types of clouds use the same underlying hardware, but differ in the service provided. The first one provides scalable computing *instances* that can be configured to serve even complex requirements. Examples of scaling of computing instances are for example Amazon's EC2 (Elastic Cloud) Services [39] and Microsoft's Azure [40], where certain sized computing instances have a certain pricing per hour. This approach is seen as the more traditional form of cloud computing. The latter one means scaling to support data- or compute-intensive applications. The *capacity* in this context means specialized applications running in the cloud that are tailored especially to intensive computations. An example of such applications is Google's MapReduce [41] which splits large tasks, such as matching text in documents, into smaller pieces and then performs the computation in parallel. This is basically the same as what cluster computing has been used before.

2.6.1 Layers of Cloud Computing

Cloud computing service providers offer a range of different cloud services to customers. As already mentioned before, the *abstraction* is an important feature of clouds. Every developer does not need the same level of detail when developing their services and applications. Thus, cloud computing allows the developing company to concentrate on their own area of expertise by dividing the offered cloud services into different layers.

The layers of cloud computing are split into three different parts, each offering a certain level of abstraction. These layers are known as *Software as a Service* (SaaS), *Infrastructure as a Service* (IaaS) and *Platform as a Service* (PaaS). The layers are shown in Figure 4.

Software as a Service (SaaS) is the most easily understood and offers the highest level of abstraction of all the layers. The software is placed in the cloud and the software developer is not required to worry about the underlying implementations, such as configuring and maintaining operating systems. Everything under the hood of the application are taken care by the cloud service provider. The services offered at SaaS layer can be considered as 3rd party hosted and maintained applications [36]. Well-known examples of SaaS applications are Google's Gmail, DropBox and SalesForce.

Platform as a Service (PaaS) layer offers the application developers more customization and control over the services used. The PaaS layer often provides the services in premade stacks, such as computing resource for software and storage services for saving data. Because of offering additional services instead of only supporting the application deployment, PaaS layer can be considered as an extension to SaaS [37]. A good example of PaaS is Microsoft Azure, which offers the operating system, SQL Azure and regular storage space all bundled together.

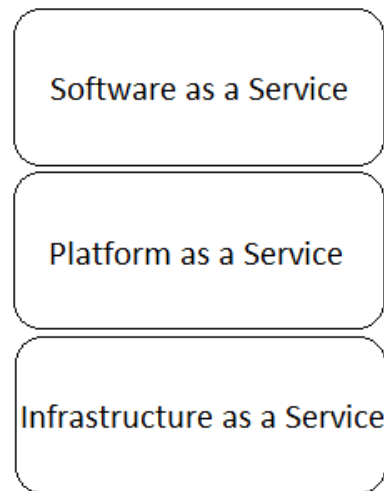


Figure 4- Layers of Cloud Computing

Infrastructure as a Service (IaaS) is the whole customizable IT infrastructure as in traditional server combinations. IaaS contains all the computing power, network resources, storage and software elements combined into a customizable package [37]. The service can be virtual machine with an operating system, an empty storage or a virtual machine with no operating system. The developer is given complete control of all the resources, while freeing the developer from maintaining the hardware. The most famous of IaaS providers is the Amazon EC2 and Amazon S3 platform for running and storing any applications.

These three layers of service can be roughly compared to their equivalents in traditional computing [36]. Figure 5 demonstrates the rough equivalences of the different layers of cloud services. The topmost SaaS layer is equivalent to the regular software layer on any computer. The Platform as a Service layer is comparable to middleware layer of a computer. The lowest of the layers, the IaaS is comparable to the operating system of a computer.

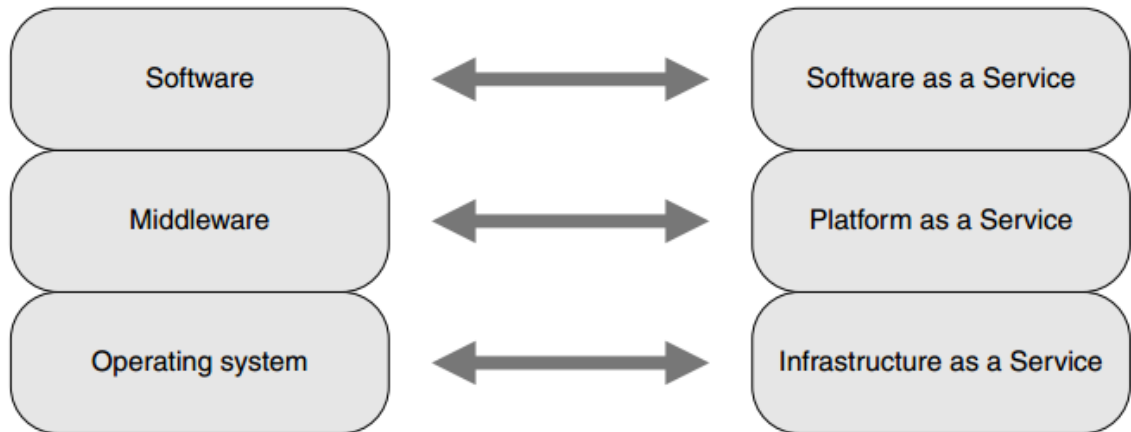


Figure 5 – Cloud computing layers compared to traditional computing layers

2.6.2 What do I need to consider when using cloud computing?

Cloud computing is currently a big buzzword that sometimes may seem as a solution to every problem. However, this is rarely a truth in real world. There are multiple aspects that need to be taken in to account when considering the usage of cloud solutions. The most discussed issues are often the *security* and *service availability*; although other matters such as data transfer capabilities exist.

Security is a big issue when dealing with cloud computing. Once all vital information is stored in a cloud, a breach in the cloud provider systems can possibly endanger all the cloud customers and their clients as well. Because of this, both of the parties, cloud service provider and the cloud service customer, need to take good care of their security in order to prevent possible misdemeanors. However, as the security is not within the focus of this thesis, it is not covered in-depth here.

Availability of the services is another issue that requires consideration. Some companies provide services that require large amounts of computing resources and cannot have downtime. A stock brokerage service for example could not close down suddenly, preventing the users from trading their stocks. The Service Level Agreements (SLA) of the service providers often covers only 99.95%-99.99% uptime. This is due to the problems that the big providers such as Amazon [42] and Microsoft [43] have suffered from.

The SLA often dictates [44] that the provider refunds money according to lost service time. The customers are not compensated based on their real losses and the damage done to their clients. Only the resources that were not available are compensated. Thus a sudden problem of the service provider can become expensive to the company using the cloud services.

Data transfer is also a limitation of the cloud services. The high bandwidth available at a common household and at companies can still be a limitation. Data-heavy applications

that require constant transfer of information can become a problem. The cloud does offer large and scalable computing resources for processing huge amounts of data. However, it might not be possible to transfer all the required information within reasonable amount of time.

The cloud computing horizon is growing with a fast pace. The last few years number one has been Amazon [45], but the scenery will quite likely change in the coming years. For the purposes of comparison, the following pages will take a look at Amazon's offered cloud services as well as to Microsoft's Windows Azure.

2.6.2.1 Amazon Web Services

Amazon has currently emerged as the leader of cloud computing. The company provides a wide range of cloud computing services known as Amazon Web Services (AWS). The most popular of these AWS are the Amazon *Elastic Cloud* (EC2) and Amazon *Simple Storage Service* (S3). The cloud services of Amazon are spread throughout the whole world. Spreading the resources to geographically different locations allows better availability and better geographical fault tolerance for the services.

Amazon Elastic Cloud (EC2) is a cloud service which provides on-demand computing resources to the customer [39]. The customers can rent these virtual computers where they can deploy their own applications. Amazon offers a wide range of different computing instances, ranging from micro level shared instance to cluster computing and high I/O instances. Table 1 shows a comparison of the EC2 on-demand computing resources that are commonly used in applications. Amazon does provide other instances as stated above, but the more specialized ones are left out from this comparison.

The comparison shown in Table 1 explains the most commonly used instances in consumer-oriented applications. These instances are named from the small micro instance to extra-large. The purpose [46] of the micro instance is to provide service

mainly for websites and applications that require additional computing power periodically. It does not work with 2 computing instances all the time, but gives small processor time slices periodically to be used in the computation.

The other computing instances work just as promised. A medium instance has always 2 computing units available, with 3.75 Gb of RAM and 410 Gb of storage space to be used. This does not change from time to time, unless otherwise changed by the user. Amazon allows the customer to automate scaling of resources, thus providing an automated way to scale the used resources up or down according to processing needs.

Table 1 – Comparison of on-demand cloud computing resources offered by Amazon

Machine Size	CPU Cores	Memory	Disk Space	I/O Performance	Cost / hour (Windows Instance)
Micro	Up to 2 EC2 Compute Units (for short periods of time)	613 Mb	None (EBS storage only)	Low	0.0035 \$
Small	1 EC2 Compute Unit	1.7 Gb	160 Gb	Moderate	0.115 \$
Medium	2 EC2 Compute Units	3.75 Gb	410 Gb	Moderate	0.230 \$
Large	4 EC2 Compute Units	7.5 Gb	850 Gb	High	0.460 \$
Extra Large	8 EC2 Compute Units	15 Gb	1 690 Gb	High	0.920 \$

The *Amazon S3* (Simple Storage Service) is an online web storage placed in the cloud [47]. It can be used for content storage and distribution, data analysis storage, backups, archiving and so on. Basically, it can store everything that a regular computer could. All the storage data is scalable in the same sense as the Elastic Cloud counterpart. The amount of data storage space can be increased and decreased depending on the customer needs.

The S3 stores customer information in blocks of storage units called buckets. One bucket can hold up to 5 terabytes data. The buckets can also be mounted to Elastic Cloud file system, in order to serve the applications deployed to Amazon EC2. A comparison of the pricing of Amazon S3 is shown in Table 2.

The storage in S3 is offered in two different categories – standard storage and reduced redundancy storage (RRS). The RRS is less expensive than the standard counterpart, but as the name explains, the data is not reproduced (backed-up) in as many places as with standard storage. The RRS is designed for storing non-critical data and information that is easily reproduced such as thumbnails and processed data for temporary storage.

Table 2 – Pricing of Amazon S3 storage

	Standard Storage	Reduced Redundancy Storage
First 1 TB / month	\$0.125 per GB	\$0.093 per GB
Next 49 TB / month	\$0.110 per GB	\$0.083 per GB
Next 450 TB / month	\$0.095 per GB	\$0.073 per GB
Next 500 TB / month	\$0.090 per GB	\$0.063 per GB
Next 4000 TB / month	\$0.080 per GB	\$0.053 per GB
Over 5000 TB / month	\$0.055 per GB	\$0.037 per GB

Table 3 – Amazon Web Services data transfer pricing

	Pricing
Data Transfer IN	
All data transfer in	\$0.000 per GB
Data Transfer OUT	
First 1 GB / month	\$0.000 per GB
Up to 10 TB / month	\$0.120 per GB
Next 40 TB / month	\$0.090 per GB
Next 100 TB / month	\$0.070 per GB
Next 350 TB / month	\$0.050 per GB

The prices for Simple Storage Service are rather inexpensive, considering the fact that they are replicated to multiple places within Amazon's facilities. This provides a reliable storage service as the customer does not need to worry about maintaining hardware or backups of the vital information.

The final thing to consider in Amazon Web Services (AWS) is the data transfer pricing. Moving information from one place to another is not free of charge. This often might not be the problem, but still it should not be left without consideration. If the developed service includes transferring huge amounts of data it might become a problem.

The pricing of AWS data is shown in Table 3. All the data coming to the service is free of charge. However, outgoing service calls are not. The first gigabyte of the month is always free, but afterwards the costs start to rise. With 0.120 USD per gigabyte after the first one, moving 1 terabyte of data would cost 119.88 USD. Considering the amount of

information transferred, the price might not be the problem. Instead, the possibility of even transferring such amount of information might be a serious issue.

2.6.2.2 Microsoft Azure

The windows Azure platform [40] was Microsoft's entrance to the Platform as a Service (PaaS) markets. The Windows Azure was only providing virtualized Windows resources, but has now opened a wider range of services. Currently Microsoft also provides Infrastructure as a Service (IaaS) with Windows Azure Virtual Machines. The main products are Windows Azure and SQL Azure.

Windows Azure is a cloud operating system running on top of a Microsoft cluster. The operating system consists of three core components: *computing*, *storage* and *fabric*. The *computing* component focuses on web applications and the *storage* answers to scalable storage needs. The *fabric* component describes the network of computing and storage nodes in a interconnected network. The main function of the fabric is to hide the connection under a layer of abstraction. With the abstraction, the user is not required to know in great detail how the parts of the used system are communicating. The fabric provides all the scheduling, resource management, device management, fault tolerance and load balancing.

The Azure computing service is quite similar to Amazon's Elastic Cloud. Microsoft provides services with quite similar processing capabilities and pricing. The machines can be bought as completely virtual machines, or just for deploying program code in a SaaS manner.

Table 4 - Comparison of Windows Azure Computing Instances

Cloud Services Instance Size	CPU Cores	CPU Speed	Memory	Instance Storage	I/O Performance	Cost/Hour
Extra Small	Shared	1.0 GHz	768 MB	20 GB	Low	\$0.02
Small	1	1.6 GHz	1.75 GB	225 GB	Moderate	\$0.12
Medium	2	1.6 GHz	3.5 GB	490 GB	High	\$0.24
Large	4	1.6 GHz	7 GB	1,000 GB	High	\$0.48
Extra Large	8	1.6 GHz	14 GB	2,040 GB	High	\$0.96

Table 4 presents the pricing comparison of Azure computation. The biggest provided machine is Extra Large, with 8 processor cores and 16 gigabytes of RAM. The offering is quite similar to Amazon's, only differing in the pricing with few cents per hour. One thing to notice with Azure is that the Service Level Agreement [44] of Windows Azure for 99.95% uptime requires the customer to rent at least two similar units at the same time. This brings the price for the most powerful virtual machine to 1382.40 USD per month. For comparison, a similar off-the-shelf 8-core computer bought from a store [48] would cost a single time fee of 1304 USD. However, this price does not include administration personnel fees nor provide any guaranteed level of service.

The SQL Azure, which provides database storage service, differs from the storage offered by Amazon. The SQL database is designed especially for structured data storage, which is normally saved in server databases. The prices for such database services are more expensive than the regular storage, as is shown in Table 5. The price of a tiny, less than a 100 Mb database is 5 US dollars per month. Going up from there,

the costs of database size drop notably. When dealing with storage sizes of over 50 Gb, the price for an individual gigabyte is less than 1 dollar per month. However, it has to be kept in mind, that Microsoft handles back-ups of the SQL databases, which is not free when done individually within the company.

In addition to SQL database storage, Windows Azure provides storage services that are equivalent to its Amazon S3 counterpart. The data is stored in same manner to storage units called blobs. These blobs are equal to Amazon's buckets and can also be mounted to virtual machines. The only difference between Azure and Amazon is the pricing of data redundancy as is shown in Table 6. Locally redundant data is stored only in one cloud service center, while the geographically redundant is stored into a second storage center within the same region.

Microsoft has also a price for data transfer going out from the cloud. For each starting gigabyte that is sent from the cloud there is a small fee to pay. Table 7 shows the pricing details of outgoing transfers. Windows Azure has two different transfer prices, 12 cents to North American and European regions and 19 cents per gigabyte to Asia Pacific region. The prices of outbound data transfer to Europe and USA are equal to Amazon's prices; the only difference is transfer to Asia, but this will likely change in the future, if Microsoft wants to compete with growing Asian markets. However, as already mentioned with Amazon's data transfer comparison, the price of the data transfer rarely becomes an issue, but is a matter that should be considered as well.

Table 5 – Listing of Windows Azure SQL pricing

DATABASE SIZE	PRICE PER DATABASE PER MONTH
0 to 100 MB	Flat \$4.995
Greater than 100 MB to 1 GB	Flat \$9.99
Greater than 1 GB to 10 GB	\$9.99 for first GB, \$3.996for each additional GB
Greater than 10 GB to 50 GB	\$45.954 for first 10 GB, \$1.998 for each additional GB
Greater than 50 GB to 150 GB	\$125.874 for first 50 GB, \$0.999 for each additional GB

Table 6 – Windows Azure data storage price listing

Storage Capacity	Geographically Redundant	Locally Redundant
First 1 TB / Month	\$0.125 per GB	\$0.093 per GB
Next 49 TB / Month	\$0.11 per GB	\$0.083 per GB
Next 450 TB / Month	\$0.095 per GB	\$0.073 per GB
Next 500 TB / Month	\$0.09 per GB	\$0.063 per GB
Next 4,000 TB / Month	\$0.08 per GB	\$0.053 per GB
Next 4,000 TB / Month	\$0.055 per GB	\$0.037 per GB

Table 7 – Windows Azure outbound data transfer pricing

DATA TRANSFER OUTBOUND	europe & usa	east asia, southeast asia
First 10 TB / Month*	\$0.12 per GB	\$0.19 per GB
Next 40 TB / Month	\$0.09 per GB	\$0.15 per GB
Next 100 TB / Month	\$0.07 per GB	\$0.13 per GB
Next 350 TB / Month	\$0.05 per GB	\$0.12 per GB

2.7 Choosing the right approach

All of the aforementioned distributed approaches have their advantages and are best suited for certain approaches. Grid computing is a geographically distributed solution, which takes advantage of computing resources across the globe. Grid is often used in research projects, to handle large and complex calculations all around the world. An energy company working on a national level does not have access to such resources nor has resources to build their own grid infrastructure. Thus, such an approach is not available to Finnish energy companies.

Multi-core computing and cluster computer are possible approaches for solving performance problems. A single multi-core computer often is not enough, as it is limited to four or eight processor cores, which might not be enough for complex processing tasks. Cluster computing on the other hand can be used to solve more complex computation problems. Clusters require more knowledge from the organizational side and good administrative personnel in order to maintain the systems. For organizations that already have existing cluster solutions and well educated staff, using clusters is a solution to consider when dealing with complex computations.

Cloud computing is scalable and inexpensive approach for organizations without existing server solutions and administrative personnel. Cloud service providers allow the deployment of applications to their hosted and maintained servers, removing the need for maintaining the hardware from the service creator side. The scalability of the cloud allows organizations to answer to changing customer demands more swiftly.

For the purposes of evaluating energy invoice computation, cloud computing is chosen because of its scalability and the abstraction from the hardware level. However, cloud computing is evaluated more in the later chapters, based on implementation requirements for the customer energy invoice calculations.

3 SYSTEM DESIGN

The design of the customer energy consumption metering is a vital part of the project. The way the system is designed and implemented brings certain requirements and restrictions to the table. The location of the storage and processing units and data transfer limitations all need to be taken into account when designing the system.

This chapter presents the invoice calculation system which is used to calculate all the customer energy invoices. The architecture of the system is examined from two different points of view: *End of the month* processing and *on-demand* processing.

The *end of the month* scenario describes a situation where all of the customer consumption is calculated only once a month in a big batch. This is to portray the situation where an energy company prepares the invoices of all the customers at once, in order to bill them correctly. The scenario brings notable amount of computation to the table at once, requiring the system to perform a large number of computations in a short amount of time. All one million customer energy consumptions are calculated together according to their energy contracts and hourly energy market prices. This whole process must be done within a couple of days, as companies require money to function.

The *on-demand* processing scenario describes a situation where a customer accesses the system over the Internet and asks for his/her current invoice situation. This service must be provided to the customers according to the government act [1]. In such a situation, the system queries the energy metering database and calculates the invoice of the ongoing month for the user in question. This process should be quite responsive in order to provide a good level quality of service. The calculated invoice that is created during the month should be stored for later use in order to prevent redundant computations when handling the end of the month computations for creating the bills for all of the customers.

3.1 Smart Metering

The smart meters are installed at the customer households by the energy companies. The meter is connected to the house's electricity center and it monitors all the energy consumption within the household. The meter gathers the consumption-related information and sends it to the energy company over the Internet. The data is then saved to the company databases, from where it will be retrieved later for calculating the customer invoice. An overview picture of is shown in Figure 6.

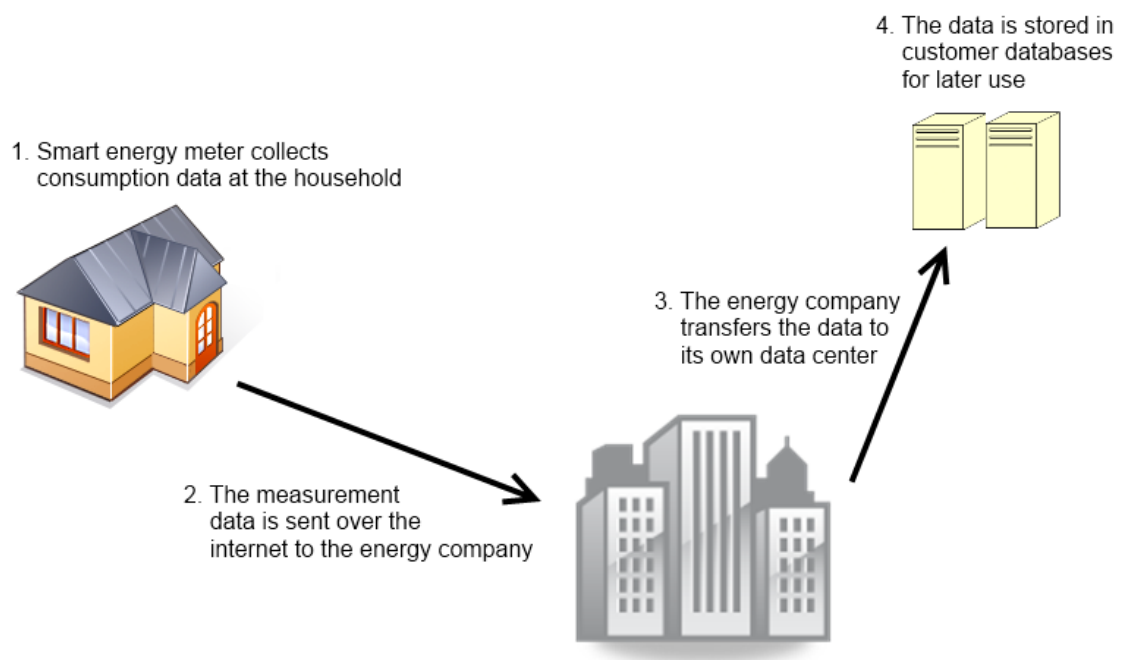


Figure 6 – The process of transferring smart energy information from customer household to company databases

Table 8 – Values received from smart electricity meters

Entry	Value	Explanation
DateStart	15.10.2011 08:00	Start Time
DateEnd	15.20.2011 08:59	End Time
Value	1500	Measured Value
Unit	kWh	Unit Type
Reliability	Estimate	Reliability of the Reading

The smart meter information sent from the household to the company premises consists of a variety of information. The received values that are important for the invoice calculation such as the reliability of the meter readings, amount of energy consumed and the timespan of the information. These values are shown in Table 8.

The first two values, DateStart and DateEnd, limit the timespan of the received values to a certain time slot. For example, the example shown in Table 8 has value of one hour between 8.00 and 9.00 on 15th of October 2011. The next two values in the table, value and unit tell the amount of electricity consumed and the scale of the value. In the same example, this is 1500 kWh of energy consumption. The final value reliability tells how reliable the reading was. Sometimes the energy meter may not be able to transmit the information to the company database successfully and can become marked as missing [48]. Sometimes, the reading can also be just an estimate, due to some meter-related issues. Thus the meter reliability has to be considered when designing the system, as faulty readings need to be retrieved again in order to calculate the customer invoices correctly.

3.2 System Composition

The invoice calculation system is composed of multiple elements. The energy companies have multiple sources of data, ranging from the customer energy meter measurements, to energy pricing contracts and customer related personal information.

All the data and system elements that are required in the invoice calculation process are shown in Figure 7.









Element	Symbol	Description
Customer		This portrays the end-user who is interested in knowing the energy consumption and the bill of the current month.
Customer Interface		The customer interface works as a mediator between the client and the service. This can be for example a web-site maintained by the energy company. Requires individual customer id.
Customer Db		The user database contains the user authentication credentials. This is not a vital part for the description of the solution in question, but it is important to keep in mind the need for user security in the final service.
Metering Db		The metering database contains all the meter information of the customer. The metering database has all the metering data, stored as one entry for each time unit (e.g. readings/hour). This data is used by the system to calculate the total consumption and invoice information.
Pricing Db		This database contains the pricing model information that is used to bill the customers. There can be different pricing models for different contracts and different moments of time (day electricity – night electricity). This information is used as the basis for calculations; it is not modified or changed by the service.
Invoice Db		The invoice database portrays a temporary data storage that contains the users' monthly invoice information. This database is used to store the results calculated during the month in order to reduce amount of repetition done.
Computing	 	This part is used to present a unit that handles the calculation of the data. Computing can happen for single customer or as a batch of customers. Computation may happen in local server or in a cloud service.

Figure 7 – Explanations of the invoice computing system elements

The elements themselves can be thought to exist in three different parts; *customer related information*, *control* and *computing* as shown in Figure 8. The *customer related information* contains all the stored data in the form of databases. This means the customer information, metering results from the smart meters, pricing information of existing contracts and calculated customer invoices. Basically all the customer related information is stored in one place, usually within the company premises. The company could for example have a centralized access point to the customer information, hiding the implementation of multiple databases.

On the other side is the *computing* element which is in charge of doing the invoice-related computations. It receives all the customer information from the company storage and uses it to calculate the proper invoice for each customer. In practice it could be any combination of computers, whether it is a cluster of a kind, cloud service or just one computer crunching numbers.

Between these two elements is a *control* element, which handles the communication between the two services. The main role of the control unit is to orchestrate the process, retrieving certain customer related information from the storage and passing it to the computing element. The customer itself does not directly access the control unit, but uses an interface provided by the energy company. The customer interface requests for certain information from the control unit that handles all the logic, returning the results to the customer to view. These elements are shown in Figure 8.

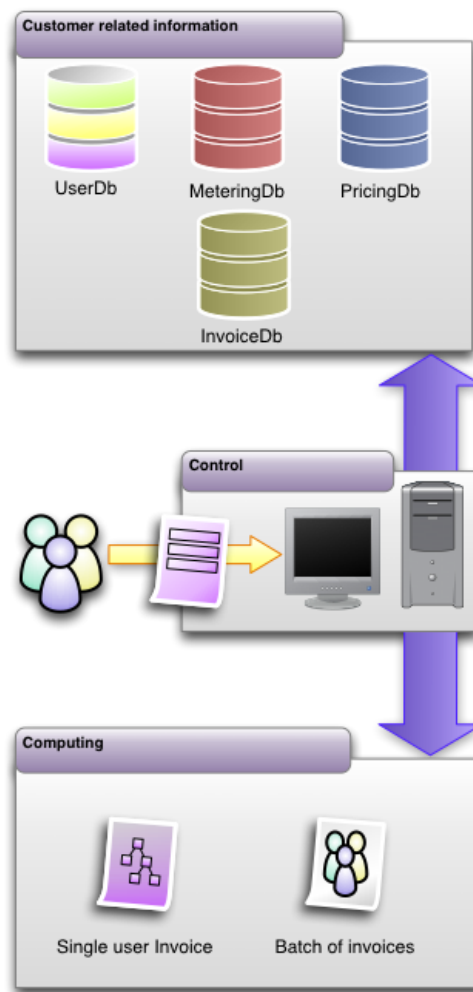


Figure 8 – Elements of invoice computing

3.3 Proposition for Computation Scenarios

The use-case scenarios of the system are split into two parts, as already explained before. The characteristics of these two scenarios pose different requirements to both of them. On one hand, the single invoice, on-demand calculation should be done within seconds after the user has requested for his consumption information. On the other hand, the end of the month calculation of all the customer invoices in one batch requires a powerful processing unit which can perform the task in less than a day. However, this spike of processing requirement only comes during certain dates and having powerful

resources standing idle most of the month is not the optimal solution. Thus an amount of scalability is required to answer to changing processing requirements.

3.3.1 Single Invoice Computation Locally

In the single invoice computation scenario, both the customer related information and the computation are located in the same place, within the company premises. As the invoice calculation is required to be done quickly, the transfer time is almost completely eliminated by placing the computing unit close by. In addition, the single invoice calculation scenario is rather simple procedure and does not require powerful computation resources. Thus, it is quite self-explanatory to say that locally placed processing unit is the best solution for the problem. The high level architecture of the single invoice computation scenario is shown in Figure 9.

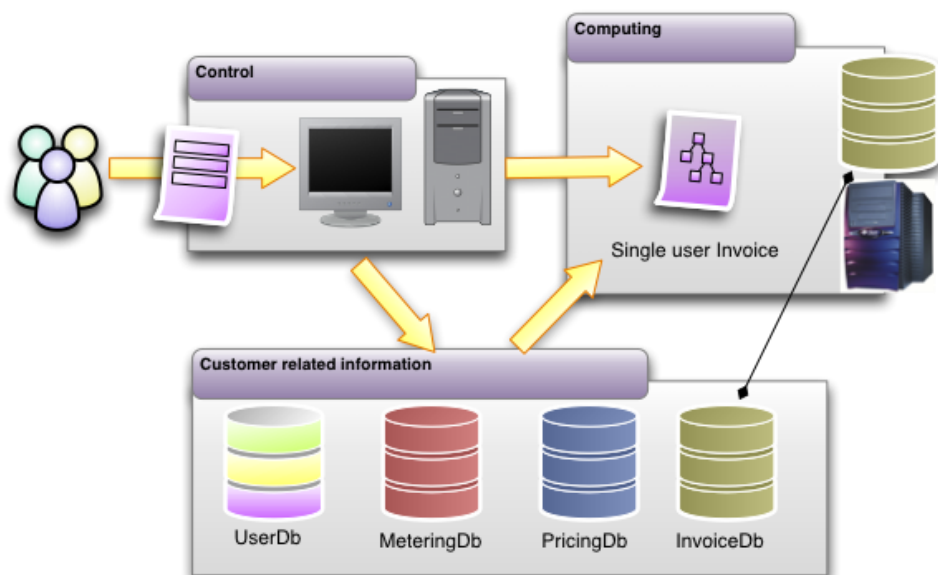


Figure 9 – Suggestion for on-demand invoice computation locally

3.3.2 Batch Invoice Computation in the Cloud

In the batch computation scenario, the processing element is placed in the cloud. The usage cloud service allows the scaling of computing resources according to the needs of the system. In this scenario the customer related information is stored within the company premises and the data required for the invoice computation is transferred over the Internet to the processing unit.

The invoice database is kept in both locally at the company as well as in the cloud. If intermediate invoice results are calculated during the month, they are stored close to the processing unit in order to lessen the amount of required data transfer. The proposed batch computation scenario is shown in Figure 10.

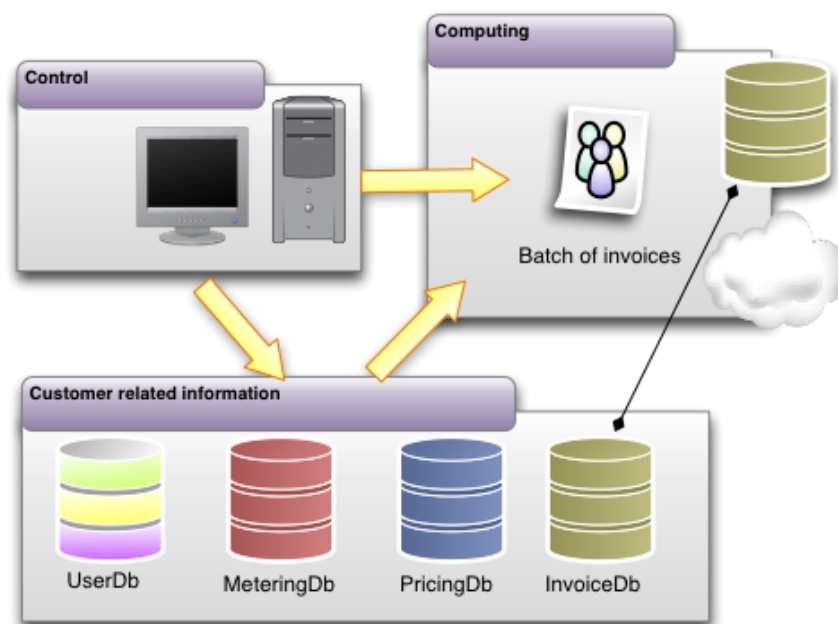


Figure 10 – Suggestions for end of the month invoice computation in the cloud

3.3.3 Evaluating Cloud Computing in invoice calculation scenarios

In order to use cloud effectively, some evaluation of the required processing, storage and data transfer amounts need to be analyzed. The complexity of the data to be processed relates to the required processing power of the system. The more computations need to be done, the more powerful computational resources need to be purchased. In addition, transferring large amounts of data may become an obstacle here as well. As the smart energy meters at customer households are generating measurement data every hour, the data transfer requirements need to be taken into account.

To start with, we need to analyze the amount of data transfer required for customer invoice computation. The data used to calculate one time unit entry is shown in Table 9. This data is based on early concept definition by the energy company and is the bare minimum information that is required in the calculation. The byte size shown in the table is based on Microsoft's data size definitions [49] for C# programming language, which is the chosen language for the implementation. As the values shown in the table are only estimates, it is quite likely that the data required in the final version of the system will require more information than is used in these simulations. In addition, the byte size does not match the real space requirements in the SQL database, as the low-level implementation is more database specific. However, the data size is related to the data transfer and how much space needs to be reserved when sent over the Internet.

The values shown in Table 9, sum up to 66 bytes per one time unit entry. It does not seem as a large amount of data. In order to understand the magnitude of data transfer required, we need to calculate the amount of data transfer that is required per user each month.

Table 9 – Byte size of the parameters required for invoice computation per time unit entry

Parameter	Parameter Type	Example value	Size (in bytes)
MeteringId	Int32	1	4
Type	String	TimeFrame	9
StartDate	DateTime	15.1.2012 12:00	8
EndDate	DateTime	30.8.2012 7:59	8
Value	Decimal	803 547	16
Status	String	Ok	2
Unit	String	kWh	3
Money	Decimal	55 874	16
Total byte size			66

Table 10 presents the proportion of the required data transfer capability. When the meter values are received every 15 minutes, it already creates 2880 entries per month which adds to 190 kilobytes in data. This is just the transfer size per user. With the scale of one million users, the monthly data transfer grows up to 190 gigabytes per month as can be seen from Table 11.

Depending on the available data connection speed, a data transfer amount of 190 gigabytes may become a problem. Uploading information to Amazon's and Microsoft's cloud services is free of charge and the only charge in transferring data to cloud is related to agreements made with the Internet Service Providers (ISP). However, transferring such an amount of data requires time. A comparison of time requirements for transferring all the customer information in one month for both 60 minute timeframe and 15 minute timeframe scenarios is shown in Table 12.

Table 10 – Required data transfer per user

	Entries / day	Entries / month	Kb / month
15 min timeframes	96	2880	190.08
60 min timeframes	24	720	47.52

Table 11 – Total data transfer size of all customers in gigabytes per month

	300 000 customers	500 000 customers	1 000 000 customers
15 min timeframes	57.024 Gb	95.04 Gb	190.08 Gb
60 min timeframes	14.256 Gb	23.76 Gb	47.52 Gb

Table 12 – Days required in transferring monthly customer data

	1 Mbit/s	2 Mbit/s	5 Mbit/s	10 Mbit/s
47 520 Mb	4,40 days	2,20 days	0,88 days	0,44 days
190 080 Mb	17,60 days	8,80 days	3,52 days	1,76 days

A high speed outbound connection speed of 10 Mbit/s, transferring 190 gigabytes takes less than two days. However, transferring the same data on a slower connection of 1 Mbit/s takes almost 18 days. Based on the transfer time requirements shown in Table 12, it can be stated that the available internet connection speed plays an important role in deciding whether to use cloud computation or not.

Another thing required for the evaluation of system requirements is the computational complexity. The computational requirements define how powerful processing units are needed for calculating the customer invoices. This information can be used, in unison with the data transfer requirements, to decide the implementation environment for the system.

Table 13 – Monthly computation operations required per client with 15 minute measurement intervals

Clients	Operations (Millions)
100 000	576
300 000	1 728
500 000	2 880
800 000	4 608
1 000 000	5 760

The amount of processing operations required is based on the additions and multiplications of the measurement data from the smart energy meters. One entry consists of the amount of consumed energy, the time of consumption and the price of energy in the energy markets at certain moment of time. These required operations per entry (o_e) are always the same for each entry. The total amount of calculations (C_T) required for one customer is calculated as shown in equation 1.

$$\text{Calculations} = C_T = d_T * e_d * o_e \quad (1)$$

$d_T = \text{days in measurement period}$

$e_d = \text{amount of entries per day}$

$o_e = \text{number of required operations per entry}$

In order to calculate the total amount of entries in one month with 15 minute measurement intervals, the amount of days in measurement period $d_T = 30$ are multiplied with the amount of entries in one day $e_d = 96$ (4 entries/hour * 24 hours = 96 entries) and this result is multiplied with the amount of required operations per entry $o_e = 2$ in order to get the total amount of operations required for one client during one month

The amount of the monthly operations per client is $30 * 96 * 2 = 5\,760$ operations. The amount of operations required with larger number of clients is shown in Table 13. Calculating the invoice for one million customer requires 5.76 billion operations. However, this should not be a problem alone, as the newest processors can achieve a computation capacity of 120 GFLOPs [49] (Giga floating operations per second). Being only straightforward calculations the problem would be less complicated. However, the operations require database access and data transfer in order to retrieve the values from multiple places and to calculate them in another. All these aspects add to the problem and affect the system's performance in a notable portion.

Implementing the invoice calculation system as a cloud solution is a valid approach. Based on the assessed requirements, the performance of the system is connected to available data transfer capabilities. A slow outgoing transfer speed will have an effect on the overall performance of the system. Thus, a fast broadband connection is required for optimal system performance. On the other hand, the amount of computation required for the customer invoice calculation suggests that the computations can also be done on a single powerful computer. Such a solution would remove the problem of transferring customer information over the internet and take advantage of the already existing on-premise computational resources. However, more research is needed in order to assess the required computational capabilities, before making any decisive conclusions. The next chapter will evaluate the system performance through simulations, in order to have more concrete results for drawing final conclusions.

4 SIMULATIONS AND RESULTS

The simulations presented in this chapter evaluate the computation time required for both *end of the month* batch processing and the *on-demand* single invoice calculations. There are quite many aspects to consider in the tests that all can affect the performance of the system. The parameters used in evaluation in the simulations are shown in Table 14.

Table 14 – Parameters used in simulations

Parameter	Values
Measurement interval	15 min, 60 min
Price interval	1/day, 2/day, 24/day
Correct estimate percentage	40%, 60%, 80%, 100%
Number of processor cores	1, 4
Number of clients	1, 100 000, 1 000 000

The first value of *measurement interval* means how often the meter readings can be received from the smart energy meters. The used values are once per hour (60 minutes) and every 15 minutes, which is considered to be a possible future measurement interval. The *price interval* means how many different pricing scenarios the customer has. One per day means that customer has a static pricing for every hour, twice per day suggest the customer having day-night electricity, where there is different electricity pricing for night and day time. The last value of 24 per day means that for every hour there is a different price tag, simulating the possibility of charging customers according to real-time energy market prices.

The correct estimate percentage is related to the amount of smart meter readings that are correct. A value of 40% means that 60% of the values are missing or estimates and are required to be updated later. This meter reliability plays an important role in the

usage of the metering tree. At the end of the month the values should be always 100% correct meaning that this value does not have effect on the batch processing.

The *number of processor cores* means how many processor cores are used for the computation. One core means doing the calculations in a sequential order, whereas four cores means distributing the computation to all available local processor cores. The last parameter of *number of clients* means on how many clients the invoice calculations are done.

4.1 Metering Tree Algorithm

An optimization algorithm was devised to be used with the invoice calculation system. The algorithm leverages the possibility of storing previously calculated customer invoices during the ongoing month and reduces the amount of required calculations later at the end of the month.

The customer consumption information is a large set of tables, each containing the amount of consumed energy and the timespan of the measurement. The invoice is then calculated by matching the energy consumption tables with customer contract tables that contain information of the agreed pricing details. For example, the customer could have night and day electricity contract, meaning that he pays a little bit higher price per kWh during the day and less during the night. This means that half of the measured entries are calculated according to one tariff and the other according to another tariff. In the situation of taxing the customer according to real energy market prices, there would be 24 different tariffs, one for each hour of the day.

With the knowledge of how the invoices are generated, it is possible to devise a tree-like algorithm for storing previously calculated data. An example of metering tree structure is shown in Figure 11. Each customer invoice is its own tree, which starts by having month as the root node. Under each month is a day node, which is then split into time frame nodes. Each time frame node is equivalent to the pricing unit in the customer's

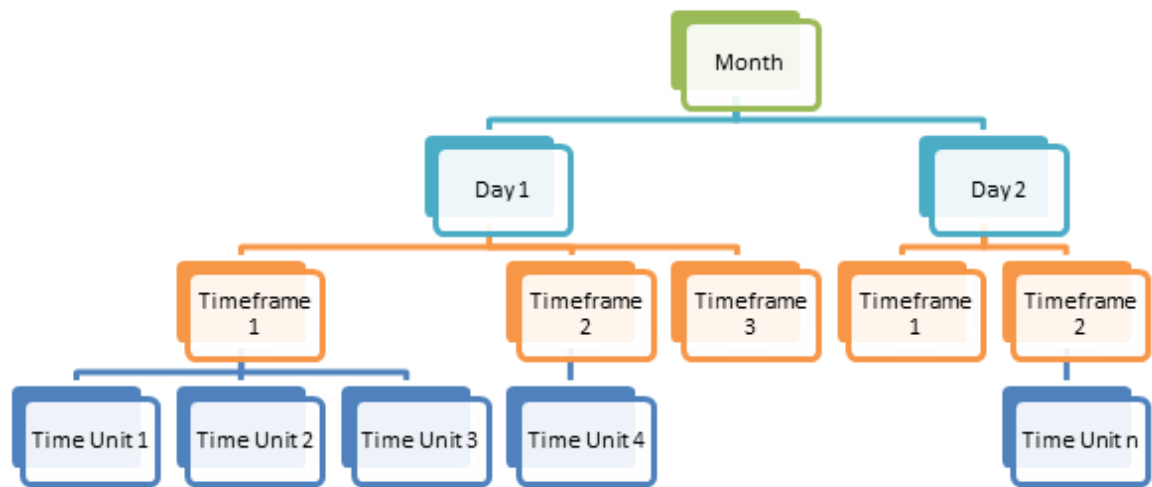


Figure 11 –An example of a constructed metering tree

energy contract. If the customer would have day-night-electricity contract, there would be three time frame nodes (00.00-07.59, 08.00-15.59 and 16.00-23.59). Each of the time frame nodes has a time unit node that is the basic time unit received from the smart meters. In current real-life situations this is one hour, but the existing meters can provide readings even every 15 minutes.

All the nodes are similar to each other. They contain information of the metering time, consumption amount, meter reliability, the type of the node (month, day, time frame, time unit) as well as the price of the electricity consumed. The values contained in a node are shown in Table 15.

Table 15 – Parameters of a metering tree node

Value	Explanation
DateStart	Start time
DateEnd	End time
Value	Measured value
Unit	Unit type
Reliability	Reliability of the reading
Money	Value in money
Type	Type of the node

4.2 Example of Metering Tree Algorithm in Use

An example of how the metering tree algorithm works is shown in Figure 12. In the example, the user logs on to the service via web interface and asks for his energy consumptions. The service retrieves the values from the company database. At moment 1, the first two time units in the database are marked as OK, meaning that they are correct. However, the third time unit is marked as an estimate. The values within this entry area not correct due to meter-related problems. The system calculates the consumptions and informs the customer of the current bill, notifying the client that the invoice is just an estimate. All the values are stored to the temporary invoice database, from where they can be retrieved later.

Few hours later, the customer returns to the service to check his current readings. The system notices that the user has already asked for the information before and retrieves the previously stored values. As one of the time units is marked as an estimate, the system accesses the metering database in order to find if the information has changed. In addition, the system retrieves the missing hours from the same database. As can be seen in moment 2, the updating of the estimated value turns out to be successful as the meter readings have changed to correct during the past few hours. The other new values are correct as well.

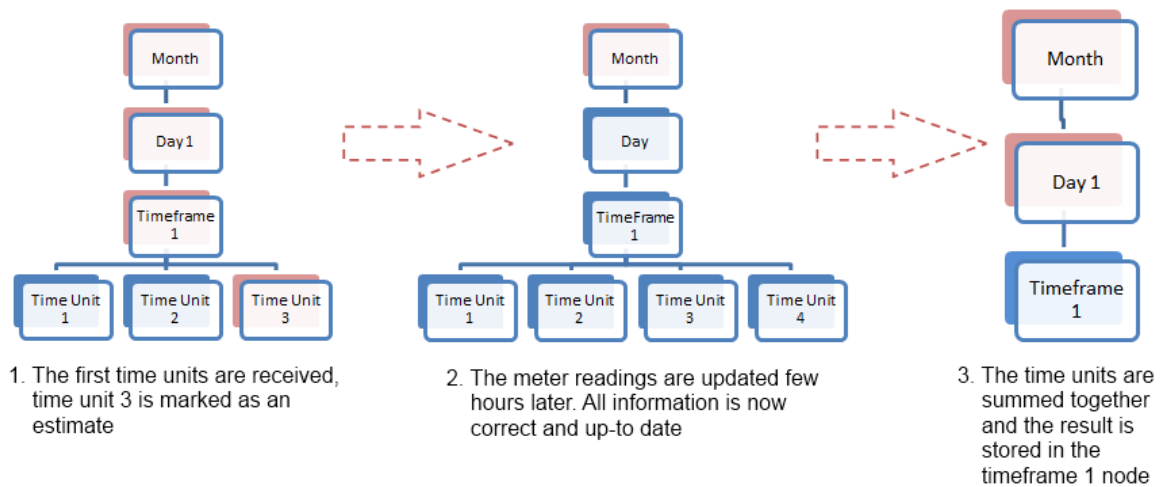


Figure 12 – Example of pruning the metering tree

Once all values under the time frame 1 are marked as OK, they can be summed up and the newly attained information can then be saved to the time frame node. This allows discarding of the nodes underneath the time frame node as they are no longer needed. Now, in moment 3, only the month, day and time frame node are saved to the temporary database. Once the user logs in next time to ask for his invoice status, there is no need to calculate previous values again, just to retrieve the time frame node. This approach saves both storage space and processing time, as the tree is pruned into a minimal possible size after each calculation.

4.3 Test Environment

The described tests were run on an off-the-shelf computer that is by no means optimized for the task. This is to portray the power of regular computer and its behavior in a server-like use. The computer has an Intel Core i7-2600 quad-core processor that runs at clock speed of 3.40 GHz. The machine is equipped with 8 Gb of RAM and is running 64-bit Windows 7 operating system.

The programming environment used in the tests uses Microsoft's .NET framework. The parallel processing used in the tests was done by using the managed threading [50]

provided by the framework via automatically managed task pooling for the sake of simplicity. As the system in question does not yet exist and all the details of the final production environment were not known, it is not feasible to delve deep in the fine grained design of threads. The automatic parallelism is in no manner perfect and manually managed threading would definitely optimize the system even further.

4.4 Runtime Tests

The runtime tests measure the effect of different measurement and pricing intervals as well as the estimate correctness on the performance times. These tests are run by using only one processor core in order to simplify the results. The customer database was dynamically generated at the start of every run to contain the information of the client in question. No other data was stored in the database in order to remove the effects of search speed of the underlying database from the equation.

The database values were first generated for the first seven days, which the processing unit then retrieved and processed. The customer invoice information is handled and stored in the tree format, as described in the previous chapter. After processing the seven day batch, another seven days were added to the database and the new invoice values were calculated for the two weeks. This was repeated four times in total, thus simulating calculations that are done once a week to update the invoice status.

All the measured estimate values were changed to correct at the generation of the next week. 50% of those energy consumption values were changed in order to portray the updating more realistically. All of the simulations were run five times and the average of those runs was taken.

The results of the average run times with both the 15 minute and 60 minute measurement intervals can be seen in Table 16, Table 17 and in Figure 13. As can be seen from the results, the most notable factors are the interval frequency and the meter

Table 16 – Processing times of 60 minute measurements of different timeframes in seconds

	40% reliability	60% reliability	80% reliability	100% reliability
1 Timeframe 60 minutes	1.5782000	1.0620000	0.6118000	0.2624262
2 Timeframes 60 minutes	1.3084000	0.9018000	0.8596000	0.3220322
24 Timeframes 60 minutes	2.5676000	2.1696000	1.7508000	0.9956996

Table 17 – Processing times of 15 minute measurements of different timeframes in seconds

	40% reliability	60% reliability	80% reliability	100% reliability
1 Timeframe 15 minutes	8.21302122	6.87348728	5.5730798	0.3242324
2 Timeframes 15 minutes	8.37620000	6.9540000	5.5020000	0.4000400
24 Timeframes 15 minutes	10.0210000	7.9490000	5.0976000	1.0703070

reliability. Changing from once per hour to every 15 minutes introduces four times more computation.

The meter reliability on the other hand means the amount of data that needs to be retrieved again and reprocessed. The higher the energy meter reliability is, the less there are missing values. When coming across missing values, the system has to calculate an estimated invoice price for a customer and mark the entry to be retrieved again later, when the next calculations are performed. This causes the invoice system to retrieve and calculate same entries multiple times. The lower the amount of missing energy meter

values, the lower the amount of recalculations and the faster is the computation. This can be seen when comparing processing times with 100% meter reliability. The difference between 1 time frame with 15 minutes intervals and the 1 time frame with 60 minute intervals is only around 0.08 seconds per client when the readings are always correct, even though there are four times more values to be calculated.

The used pricing intervals add up to the complexity as well. Having different taxation for different hours requires slightly more computation (and the tree is built more complex as well). However, as the precision of the meter readings grow, the calculation times drop in notable proportions. With 24 time frames and 15 minute measurement intervals and 40% correct readings, the computation time is 10 seconds for one client. Once the reliability grows up to 100%, the required computation time drops down to a mere one second.

Figure 13 shows, that the more correct the meter readings are, the faster it is to calculate the customer invoice. The runtime gap drops notably when the meters can provide 100% correct information. However, in reality this is rarely the case. In order to deal efficiently with the problem at hand, more optimization than just the metering tree algorithm is required.

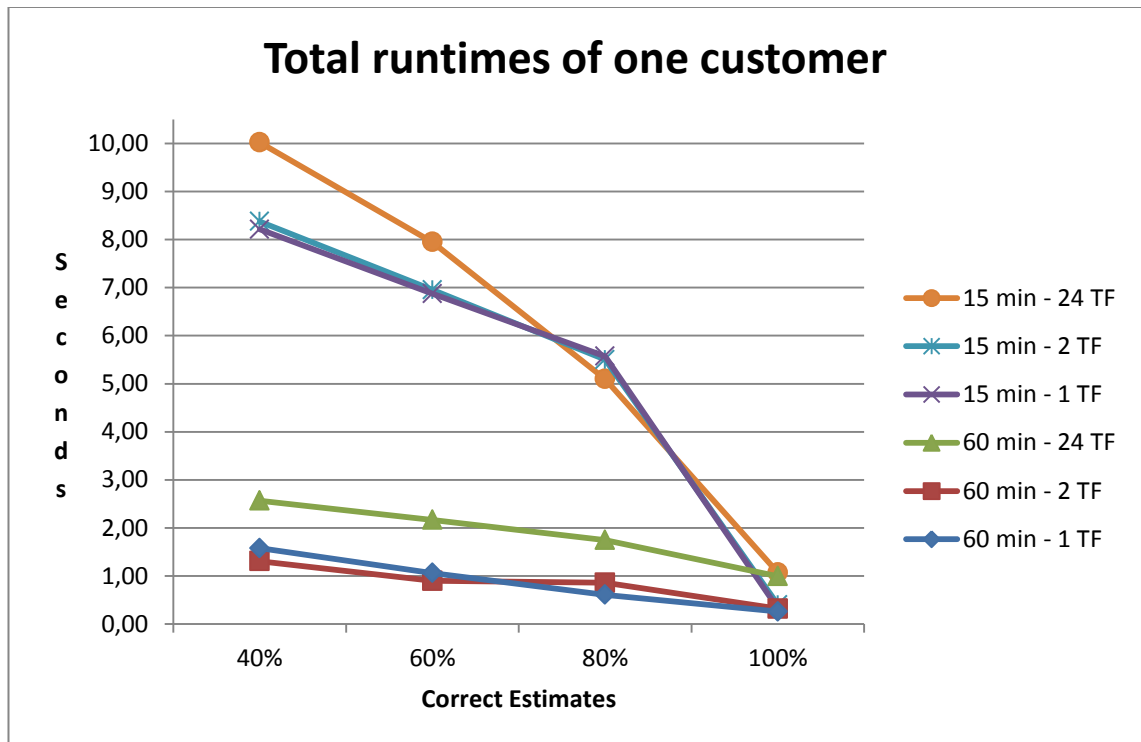


Figure 13 – Total runtimes with 15 and 60 minutes measurement intervals for one customer

4.5 Parallel Simulations

The parallel simulations takes a look at solving the computation problem by utilizing multiple processor cores. In this experiment, the calculation task is split into smaller parts that are handed out to different processor cores for calculation. When done in this manner, the full processing power of the computer can be manifested for use. The computations done here are performed on a single local machine, which was presented in section 4.3 -Test Environment.

The user invoice calculations in this simulation are instantiated as tasks. A task is the complete process that should be done to a client and is performed in one thread. Figure 14 shows the breakdown of one task. The task starts with retrieving the given customer information from the databases. In order to prevent the database from becoming a

bottleneck, more than one customer information is retrieved at once (e.g. 1000 customers) and then processed as a batch. With this method, the threads can be estimated to finish at different moments of time, making it less likely that all threads are querying the database at the same time.

1. Get Entries of x Customers
2. For x Customers
 - 2.1 Build Metering Tree
 - 2.2 Prune Tree
3. Save Results

Figure 14 – Listing of task run in parallel simulations

Once the entries are retrieved from the database, the metering trees are built from these values. After building the tree, it is pruned into a compact form as shown in metering tree example in Figure 12 and the results are calculated. Once all customer trees are built, the results are stored in one batch to the invoice database.

The most important measurement to carry out in the parallel simulation is the difference in processing time between serial and parallel computation. The computation was performed as an *end of the month* batch, where everything has to be calculated for each user from the scratch. In addition, a comparison was made against the solution of calculating the invoices in smaller portions every seven days. This simulates a situation where all the values are updated once a week, thus splitting the computational load to a longer period of time. In such scenario, only one week is calculated at the end of the month, making the final invoicing to happen faster compared to calculating all four weeks at the end of the month.

The tests were run against a database of 1000 clients. The process was split into 125 tasks, each containing 8 users that had 672 entries. The tests were run 100 times against the database, resulting in an estimate of computation time for 100 000 customers. The comparison was done with 3 different processing possibilities; 1) *serial batch*

processing with one core, 2) parallel batch processing with four cores and 3) parallel end of the month processing with four cores. The results can be seen in Table 18, Table 19 and in Figure 15.

Table 18 – Comparison of execution times in hours

Clients	1 Core batch	4 Cores batch	4 Cores – Final week
100 000	5.66 h	2.91 h	1.15 h
1 000 000	56.57 h	29.06 h	11.51 h

Table 19 – Comparison matrix of execution times of parallel and serial processing in percentages

	1 Core batch	4 Cores batch	4 Cores – Final week
1 Core batch	100.00 %	194.69 %	491.59 %
4 Cores batch	51.36 %	100.00 %	252.50 %
4 Cores – Final week	20.34 %	51.36 %	100.00 %

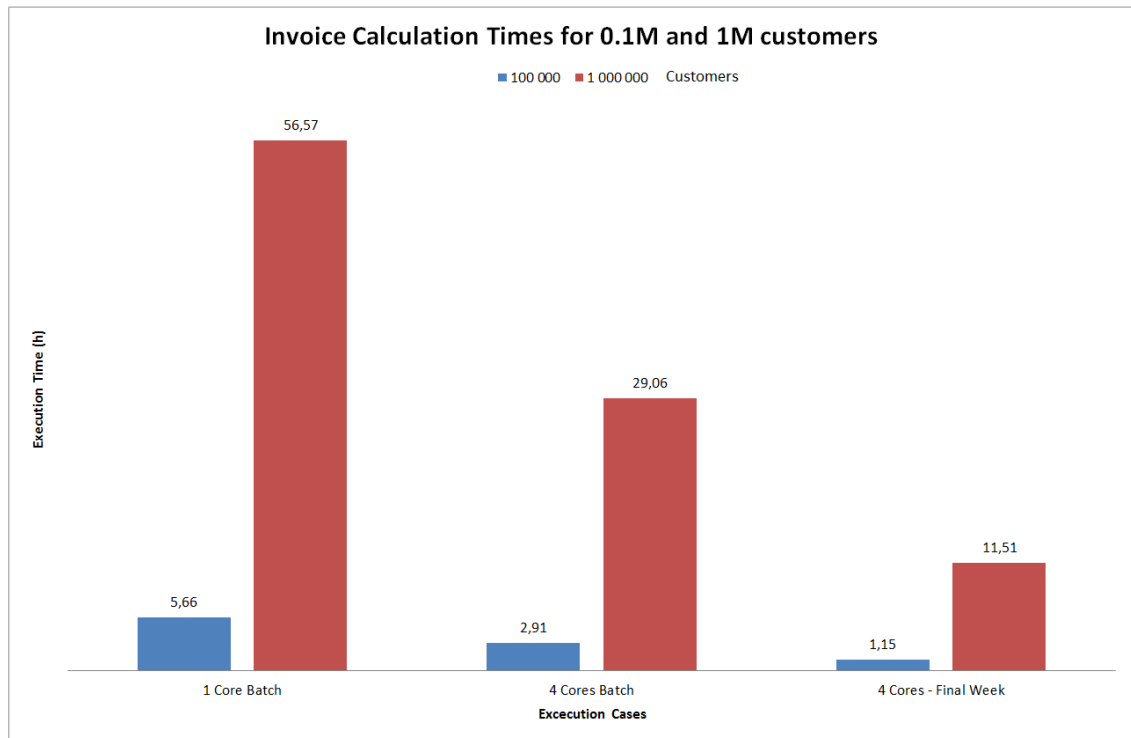


Figure 15 – Comparison of processing times in hours with serial and parallel processing

The outcome of the simulations shows that calculating 1 million clients with the traditional sequential computing would take over 56 hours to complete. A result of over two days of processing is not an acceptable result. However, when all the available processor cores are used in the calculations, the same operation takes only 29 hours, which is a drop to 51% of the original value.

Additionally, if the parallel computation is combined with the calculation of intermediary results during the month (e.g. once a week), the processing time is even faster. The last week case requires only updating of the 4th week's values, as the other three have already been calculated. This drops the required time to a mere 11.5 hours, which is only 20% of the original time.

Despite of this boost at the end of the month, this approach requires the system to calculate all the user invoices beforehand, for example once every week. However, if the processing system is owned by the company and only dedicated to the invoice computing, it is wise to balance the load evenly throughout the whole month, instead of just focusing everything to one or two days. In addition, if the customers have the possibility to monitor their own resources via the Internet, on-demand, some of the computation are already processed during the month, without enforcing.

5 CONCLUSION

When looking at the possible options for implementing the invoice computation system, the possibilities vary from Grid computing to local processing and cloud computing. This study concentrated on evaluating cloud computing and local computing as a realistic and feasible solution for the task at hand. Both cloud computing and local multi-core processing can be used in implementing the invoice calculation system.

Cloud services are scalable and can answer to fluctuating processing requirements better than traditional on-site servers. However, data transfer requirements can become a hindrance to overall system performance. With one million clients, the invoice calculation system needs to transfer 190 gigabytes of customer data every month in order to calculate the customer invoices correctly. The existing data connections can handle the transfer loads, allowing the usage of cloud computation. However, large energy companies already have their own computational infrastructures in place, which makes using own resources more preferable.

Another problem that emerges with cloud computing is the pricing of processing time. In both Windows Azure and the Amazon Elastic Cloud, the rent for decent computers outweigh the price of purchasing own computer within a few months. As energy companies have existing servers for storing and processing customer related information as well as own administrative staff, purchasing cloud computing is not currently a viable option to be considered. However, for a company that does not have their own administrative personnel and servers, cloud computing is a good solution to be considered.

The results gained in this research show a certain trend. As the data size grows to a notable proportion with the increase of metering interval frequency, the processing requirements grow as well. Calculating the clients' invoices all at once is an option that doesn't require notable development effort. However, the calculation time for one

million clients would be over 56 hours at the end of the month, when the invoice information is needed in order to bill the customers correctly. Instead of calculating the invoices in a sequential manner, partitioning the problem into smaller tasks for parallel computation brings a noteworthy change in the performance times. The simulations presented in the study show a 50% drop from the original 56 hours to a little over a day.

One approach to reducing the computation times is to calculate the customer invoices during the measurement period, e.g. once every week and save the results. By calculating the intermediate results, the processing load is spread more evenly throughout the whole month and the final results can be received faster. The tree-like optimization algorithm introduced in the thesis is an efficient solution for saving space, data transfer load and processing time in storing and updating the intermediate results. When this tree algorithm is combined with the parallel computation approach, utilizing all the available processor cores, the calculation time of 1 million customers is reduced to a mere 11 hours. This resulting time is only 20% of the original sequential batch processing.

Considering all the aforementioned facts, the best solution for an energy company would be to implement the invoice system as a local, on the premises solution. The costs of purchasing own server outmatch the costs of available cloud services currently at the market. The customer related information will then be close to the processing unit, thus reducing the data transfer problem to a minimum. Parallel processing provides a significant gain to processing speed and the service should be developed with parallelism in mind. The tests were carried out by using just one off-the-shelf computer, but in order to gain even better performance, multiple computers can be combined to work together.

In addition, giving customers the possibility of monitoring their own energy consumption and invoice information over the Internet requires on-demand computation. When this functionality is provided, the attained results should be stored in temporary database in order to decrease the required processing time later on.

6 REFERENCES

- [1] Finlex, "Valtioneuvoston asetus sähkötoimitusten selvityksestä ja mittauksesta," 17 03 1995. [Online]. Available: <http://www.finlex.fi/fi/laki/alkup/2009/20090066>. [Accessed 03 10 2012].
- [2] Cluster for Energy and Clean Environment, *Smart Grids and Energy Markets Factsheet*, 2011.
- [3] S. Hirsjärvi, P. Remes and P. Sajavaara, *Tutki ja Kirjoita*, Helsinki: Kustannusosakeyhtiö Tammi, 2008.
- [4] F. A. Veal Bryan, "Performance Scalability of a multi-core web server," in *Proceedings of the 3rd ACM/IEEE Symposium on Architecture for Networking and Communications systems, ANCS 07*, New York, 2007.
- [5] G. Moore, "Cramming More Components onto Integrated Circuits," *Electronics Magazine*, vol. 38, no. 8, 19 April 1965.
- [6] D. S. Johnson and L. A. McGeoch, "The Travelling Salesman Problem: A Case Study in Local Optimization," in *Local Search in Combinatorial Optimization*, London, John Wiley & Sons, 1997, pp. 215-310.
- [7] M. Dorigo, V. Maniezzo and A. Colomi, "The Ant System: Optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics--Part B*, vol. 26, no. 1, pp. 29-41, 1996.
- [8] O. Astrachan, "Bubble sort: an archaeological algorithmic analysis," *ACM SIGCSE Bulletin*, vol. 35, no. 1, pp. 1-5, 2003.
- [9] C. Hoare, "Quicksort," *Computer Journal*, vol. 5, pp. 10-15, 1962.
- [10] M. Baker and R. Buuya, "Cluster Computing at a Glance," *High Performance Cluster Computing: Architectures and Systems*, vol. 2, pp. 3-47, 1999.
- [11] A. Birrel, "An introduction to programming with C# threads," Microsoft Research Technical Report, 2003.
- [12] H. Sutter, "The free lunch is over: A fundamental turn toward concurrency in software," *Dr. Dobbs' Journal*, vol. 30, no. 3, 2005.

- [13] B. Clay, *The Art of Concurrency*, O'Reilly Media, 2009.
- [14] A. Birrel, "An Introduction to Programming with Threads," Systems Research Center, Palo Alto, 1989.
- [15] G. Simson, "Architects of the Information Society, Thirty-Five Years of the Laboratory for Computer Science at MIT," MIT Press, 1999.
- [16] V. Vyssotsky, "Structure of the multics supervisor," in *Fall Joint Computer Conference*, 1965.
- [17] M. F. Armbrust, G. Armando, J. Rean, D. K. Anthony, K. Randy, L. Andy, P. Gunho, R. David, S. Ariel and Z. M. Ion, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50-58, 2010.
- [18] K. Hwang, J. Dongarra and G. Fox, *Distributed and Cloud Computing: From Parallel Processing to the Internet of Things*, Morgan Kaufmann, 2011.
- [19] R. Buyya, S. Y. Chee, V. Srikumar, J. Broberg and I. Brandic, "Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599-616, 2009.
- [20] I. Foste, Y. Zhao, I. Raicu and S. Lu, "Cloud Computing and Grid Computing 360-Degree Compared," in *Grid Computing Environments Workshop*, 2008.
- [21] A. Shilov, "Intel to Boost Number of Cores Inside Next-Gen Xeon, Core Extreme Microprocessors," Xbit Labs, 01 10 2012. [Online]. Available: http://www.xbitlabs.com/news/cpu/display/20121001215924_Intel_to_Boost_Number_of_Cores_Inside_Next_Gen_Xeon_Core_Extreme_Microprocessors.html. [Accessed 04 10 2012].
- [22] P. Gepner and M. Kowalik, "Multi-core processors: New way to achieve high," in *Parallel Computing in Electrical Engineering, 2006. PAR ELEC 2006. International Symposium on*, 2006.
- [23] R. Ramanathan, *Intel multi-core processors: Leading the next digital revolution technology*, Intel Magazine, 2006.
- [24] S. Roberts and J. Akhter, "Multi-core programming, increasing performance

through software multi-threading," Intel Press, 2006.

- [25] Computer Science and Technology Board, National Research Council, and Academy Industry Program, National Academy of Sciences, Supercomputers: Directions in Technology and Applications, The National Academies Press, 1989.
- [26] Defence News, "<http://www.defencenews.in/defence-news-internal.asp?get=new&id=500>," Defence News, 30 05 2011. [Online]. Available: <http://www.defencenews.in/defence-news-internal.asp?get=new&id=500>. [Accessed 03 10 2012].
- [27] M. Baker, A. Apon, R. Buuya and J. Hai, "Cluster computing and applications," in *Encyclopedia of Computer Science and Technology*, New York, USA, Marcel Dekker, Inc., 2000, pp. 87-125.
- [28] M. Baker, "Cluster computing white paper," University of Portsmouth, UK, Portsmouth, 2000.
- [29] I. Foster, *The Grid: A New Infrastructure for 21st Century Science*, John Wiley and Sons, Ltd, 2003.
- [30] S. Teasley and S. Wolinsky, "Scientific collaborations at a distance," *Science*, vol. 292, no. 5525, pp. 2254-2255, 2011.
- [31] D. De Roure, M. A. Baker, N. R. Jennings and N. R. Shadbolt, *The Evolution of the Grid*, John Wiley & Sons, Ltd, 2003, pp. 65-100.
- [32] F. Berman, G. Fox and T. Hey, *The Grid: Past, Present, Future*, John Wiley and Sons, Ltd, 2003, pp. 9-50.
- [33] Salk Institute for Biological Studies, "Protein data bank worldwide repository for the processing and distribution of 3-d biological macromolecular structure data," [Online]. Available: <http://www.mcell.cnl.salk.edu>. [Accessed 10 5 2012].
- [34] Biomedical Information Research Network, "Biomedical informatics research network, birn grid," [Online]. Available: <http://www.birncommunity.org>. [Accessed 10 5 2012].
- [35] R. L. Grossman, "The Case for Cloud Computing," *IT Professional*, vol. 11, no.

- 2, pp. 23-27, March 2009.
- [36] O. Arasaratnam, Introduction to Cloud Computing, John Wiley & Sons, Inc., 2011.
- [37] P. Jonathan, "Introduction to cloud computing, embracing a disruptive force," University of Ottawa, Ottawa, Canada, 2011.
- [38] I. Salo, Cloud computing - palvelut verkossa, Finland: Docendo, 2010.
- [39] Amazon, "Amazon elastic compute cloud (amazon ec2)," Amazon, [Online]. Available: <http://aws.amazon.com/ec2/>. [Accessed 3 5 2012].
- [40] Microsoft, "Windows Azure," Microsoft, [Online]. Available: <http://www.windowsazure.com/>. [Accessed 3 5 2012].
- [41] Google, "Google mapreduce introductio," Google, [Online]. Available: <http://code.google.com/intl/fi-FI/edu/parallel/mapreduce-tutorial.html>. [Accessed 3 5 2012].
- [42] M. Rich, "Major amazon outage ripples across web," 2011. [Online]. Available: <http://www.datacenterknowledge.com/archives/2011/04/21/major-amazon-outage-ripples-across-web/>. [Accessed 23 8 2012].
- [43] H. Mark, "Leap day problems cripple microsoft's windows azure," 2012. [Online]. Available: <http://www.pcmag.com/article2/0,2817,2401005,00.asp>. [Accessed 23 8 2012].
- [44] Microsoft, "Microsoft azure service level agreement," Microsoft, 2012. [Online]. Available: <http://www.windowsazure.com/en-us/support/legal/sla/>. [Accessed 16 5 2012].
- [45] B. Darrow, "Amazon is No. 1. Who's next in cloud computing?," GigaOm, 14 03 2012. [Online]. Available: <http://gigaom.com/cloud/amazon-is-no-1-whos-next-in-cloud-computing/>. [Accessed 02 10 2012].
- [46] Amazon, "Micro Instances," Amazon, 15 08 2012. [Online]. Available: http://docs.amazonwebservices.com/AWSEC2/latest/UserGuide/concepts_micro_instances.html. [Accessed 03 10 2012].
- [47] Amazon, "Amazon Simple Storage Service (S3)," Amazon, 2012. [Online].

- Available: <http://aws.amazon.com/s3/>. [Accessed 03 10 2012].
- [48] A. Kingsley-Hughes, "Building an 8-core AMD FX system," ZDnet, 20 10 2011. [Online]. Available: <http://www.zdnet.com/blog/hardware/building-an-8-core-amd-fx-system/15679>. [Accessed 05 09 2012].
- [49] Energiateollisuus, "Tuntimittauksen periaatteita," 18 11 2010. [Online]. Available: http://energia.fi/sites/default/files/tuntimittaussuositus_2010_paivitetty20111118.pdf. [Accessed 10 01 2012].
- [50] Sharper Tutorials, "Data Types and Ranges," Sharper Tutorials, 09 11 2007. [Online]. Available: <http://sharperutorials.com/data-types-and-ranges/>. [Accessed 10 04 2012].
- [51] Intel, "Intel® Core i7-2600 Desktop Processor Series," 13 02 2012. [Online]. Available: http://download.intel.com/support/processors/corei7/sb/core_i7-2600_d.pdf. [Accessed 05 10 2012].
- [52] J. Richter, CLR via C#, Redmond, Washington: Microsoft Press, 2010.