Lappeenranta University of Technology

SGEM – Distributed Invoicing

Communications Software Laboratory

Janne Parkkila

Contents

1 Introduction

The goal of this paper is to research different methods for client invoicing for Empower Oy. The research carried out is part of work package 4 of the Smart Grid and Energy Markets project.

With the installation of smart energy meters that send real-time information of the client consumption to the energy companies, the efficient calculation of user invoices becomes a problem. There are two clear situations when the price needs to be calculated quite fast. The first situation is the end of the month, when all the clients' invoices need to be calculated in order to bill the clients correctly. The second case is informing the clients of their current energy consumption. If the clients are given the possibility to monitor their own consumption over the internet and show the current invoice status, the calculation has to be done quickly in order to provide a service with a good level of quality.

This paper takes look at different methods of performing the invoice calculation. Both regular serial processing and parallel processing are considered for the sake of efficiency. In addition, the possibility of distributing the computation to cloud services is researched. The paper also presents an algorithm for efficient calculation of the user invoices at certain moments of time and how to store them, in order to reduce the final computation time required.

1.1 Problem Statement

In this document distributed customer invoice computing is considered. This document considers solutions for two separate cases of invoice computing:

Batch computing that happens in the end of each month
Continuous computing that happen based on customer created events

1.2 End of the Month Invoicing

At the end of the month, the invoices for all the customers have to be calculated. Usually this is done as a batch processing, where the whole bunch of data is processed at once. In this situation, the processing speed is the most important factor, as the invoices have to be calculated correctly within reasonable amount of time. This paper takes a look to the possibilities provided by the use of parallel computing on controlling the processing capacity. Both local multiprocessor environment as well as cloud computing are evaluated. Emphasis on the part here is given to the scalability of the solution.

1.3 Event-based Invoicing

The event-based invoicing works differently from the previous situation. In this approach, the invoice is calculated on-demand, which means continuous computing. For example, if the user logs to the electric company website and requests for his current invoice information, the event is fired. This event causes the service to calculate the consumption and the invoice of the ongoing month. In this case emphasis is given to the computation algorithm so that the computations done by this time can be used in batch computing to shorten the required time.

Even though the meter readings available might not be perfect, a decent estimate can be given to the user. The research done takes a look at calculating the event-based information in a very short amount of time. Also a thought is given on how to store the results in order to save time in future calculations.

1.4 Invoice computing

The meter readings are an important part of the whole project. The meters installed in the customer's houses provide electricity consumption information at certain points of time. One meter reading is considered to be the basic time unit (for example, one hour). These time units consist of information such as energy consumption, time span (e.g. 12.03.2012 03:00:00 – 12.03.2012 04:00:00) reading reliability (ok, estimate, missing) and other such data related to the measuring location. This information is then saved to the metering database and used as the basis for calculating the client invoices. Figure 1 shows an example of possible meter readings received over a period of time.

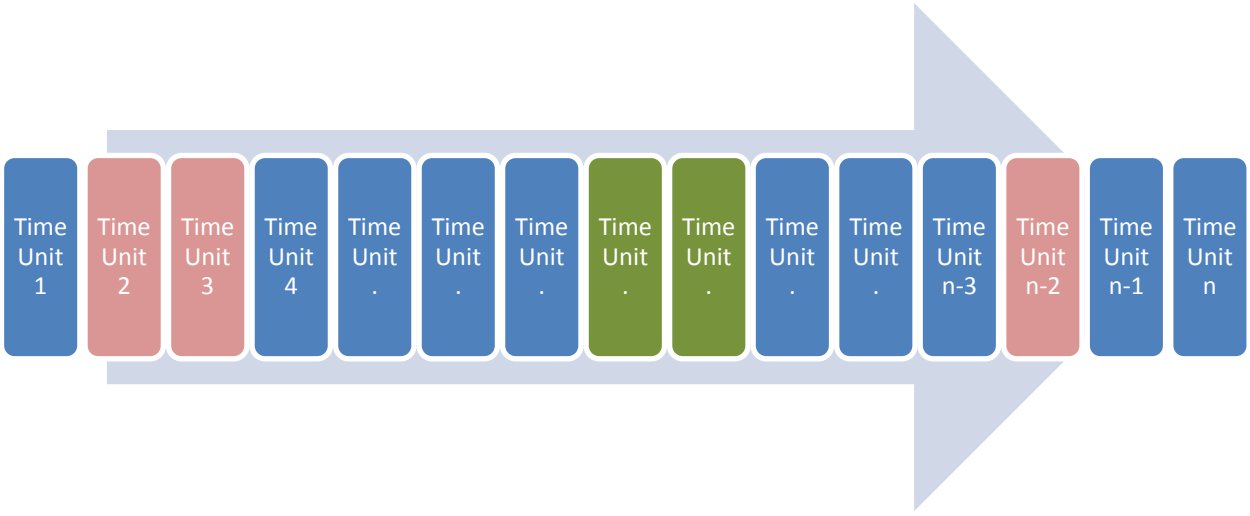

Figure 1 - Meter Readings

Color explanations:

Blue = Reading OK
Red = Estimated Reading
Green = Missing Information

As can be seen from this example, the Time Unit 1 is marked as okay and Time Units 2 and 3 are estimates. In addition, some readings are missing (marked as green) from the reading series. The reasons for having inadequate information can vary, but is usually related to the received meter readings. When the client wants to know his energy consumptions on a certain moment of time, the result might not be perfect (due to lack of meter information), but probably sufficient enough. If the client would return the next day, he could probably receive corrected information from the previous day. However, it would be waste of resources to perform all the calculations completely

from scratch each time. Saving the old results from previous day and just updating the faulty values and adding the new values accrued during the night would make a lot more sense. Thus, to save calculation time, we have devised the following proposition; creating a metering tree. The solution is explained in chapter 3 Computation optimization.

2 Overview on Architecture

This chapter introduces the architecture used for distributed invoice computing. The general architecture for the problem is given in Figure 2. *Customer related information* module contains the necessary information for invoice computing purposes. *Control module* manages the computing of the invoices by utilizing appropriate computing platforms. *Computing* module offers the available computing platforms for invoice computing purposes. *Storage* module is used for storing the results of invoices.

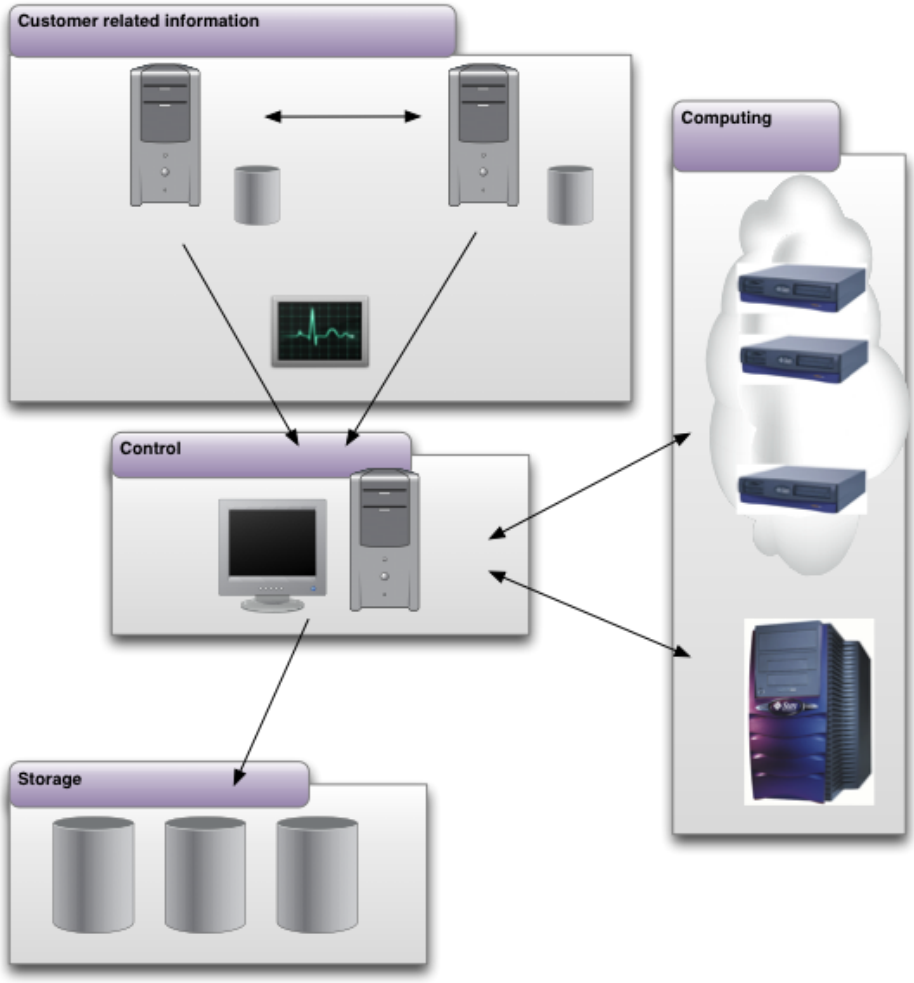

Figure 2 - General architecture for distributed invoicing

2.1 Elements used for invoice computing

The elements considered in distributed invoice computing are further described in Figure 3 and Table 1.

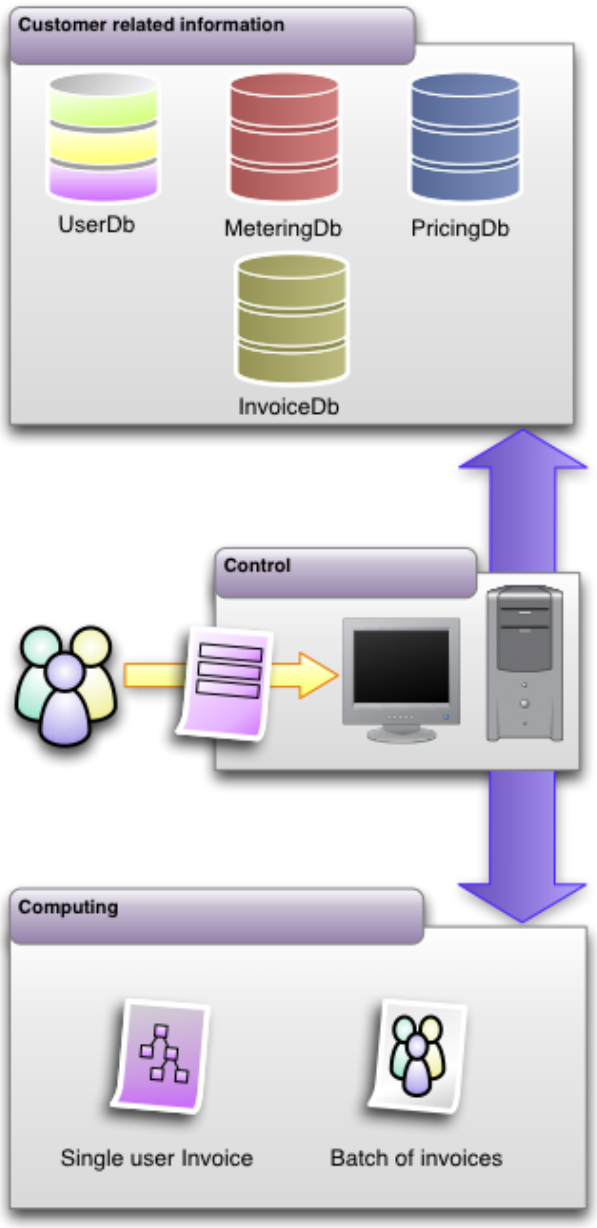

Figure 3 - Elements used for invoice computing

For the invoice computing point of view important parts are a) various customer related data sources (assumed all databases separate), b) computing algorithms and c) interface for the information (affects to the computation)

Table 1. Elements of distributed invoice computing described.

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.

2.2. Scenarios for distributed invoice computing

Various scenarios for distributed invoice computing purposes were created in the project. These scenarios took into account:

Computing environment – Local 4-core processor server and scalable cloud service
Computing algorithm – Single user invoice computing (event based) and batch based invoice computing (monthly)
Computation optimization – tree based invoice computing
Data locality – Location of various databases,

As a result the following two scenarios were selected for evaluation

Local processing scenario for event based single customer invoice computing
Cloud processing scenario for batch invoice computing

2.2.1 Case – Local Processing

The first case describes a situation where the computing module is its own server (/service). In this approach, the computing is done on a regular off-the-rack computer. The computing service would consist of the computing unit and the temporary invoice database. The temporary database holds all the calculated consumption information and resides close to the computer in charge of the calculations. The computing of the data is done within the same service, thus requiring access to the actual metering database only when new metering data needs to be fetched. This reduces the amount of required service calls to the metering database. As the already made calculations are stored in the temporary database, less computational resources are needed to calculate the updated invoice information. This scenario is presented in Figure 4 and is meant for event based processing of a single customer but can be tested for batch processing as well.

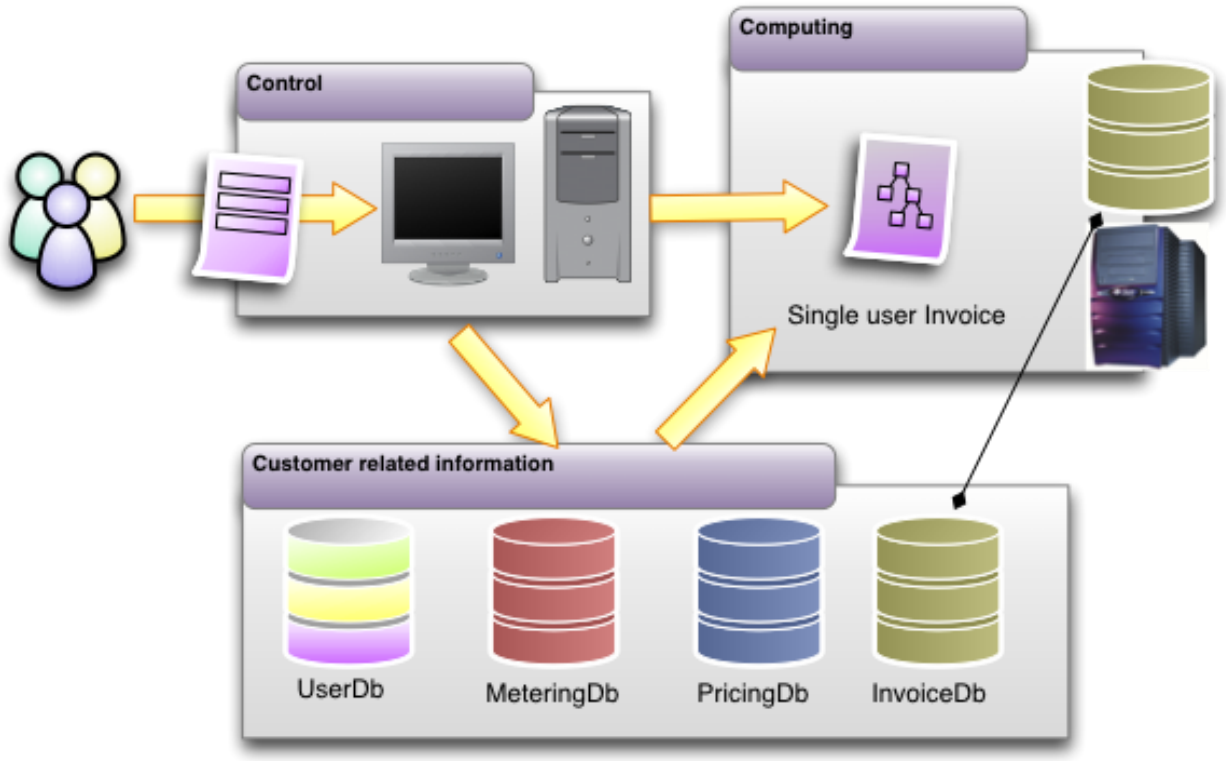

Figure 4 - Local processing scenario

2.2.2 Case –Cloud Processing

The second case describes a situation where computing is performed on openly available and scalable cloud environment. As in local case customer related information is located in their original locations and only temporary invoice database is kept with the computing module. This scenario is meant for batch processing purposes. This requires separate data transmission between computing module and databases as batch computing needs the data for invoice computations. The data transmission time needs to be considered separately for this scenario.

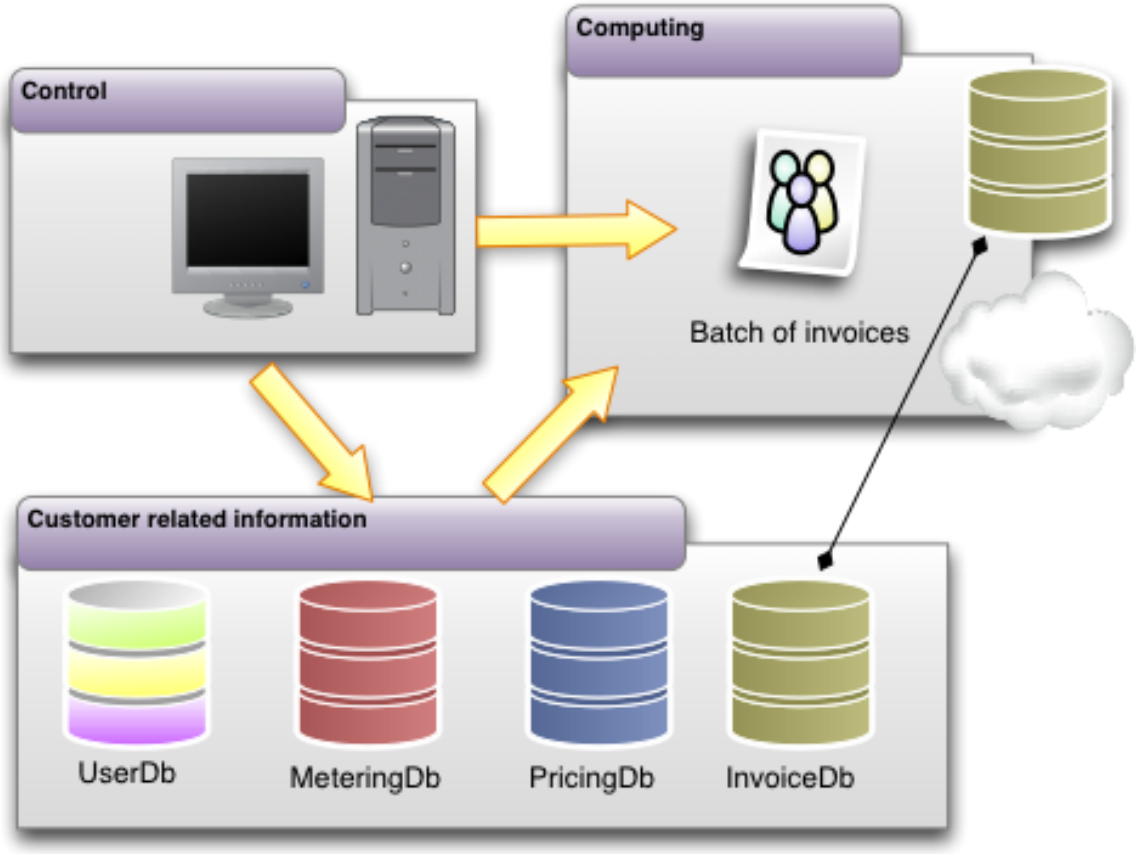

Figure 5 - Hub Storage and Cloud Processing

3 Computation optimization

This chapter presents an approach for invoice computing suitable for both single customer and batch computing.

3.1 Tree based computing

The idea of the tree based computing is to split the monthly meter readings to a tree-like format for temporary storage. The tree contains the information of the usage place in question (although it can be any logical unit that best describes the readings, e.g. client id). Each of the nodes contains the same information that is partially inherited from the metering, time period and invoice databases. Basically they hold the amount of electricity consumed, the price and the time span. All the nodes are split logically, the Month being the top of the tree. Underneath are all the days of the month and under each day is a certain timeframe. The timeframe depends on the existing time periods. For example, the client might have night and day electricity, which would split the day into three time frames (00-08, 08-20, 20-24). Each time frame is then divided into hours. An abstraction of the structure is shown in Figure 6.

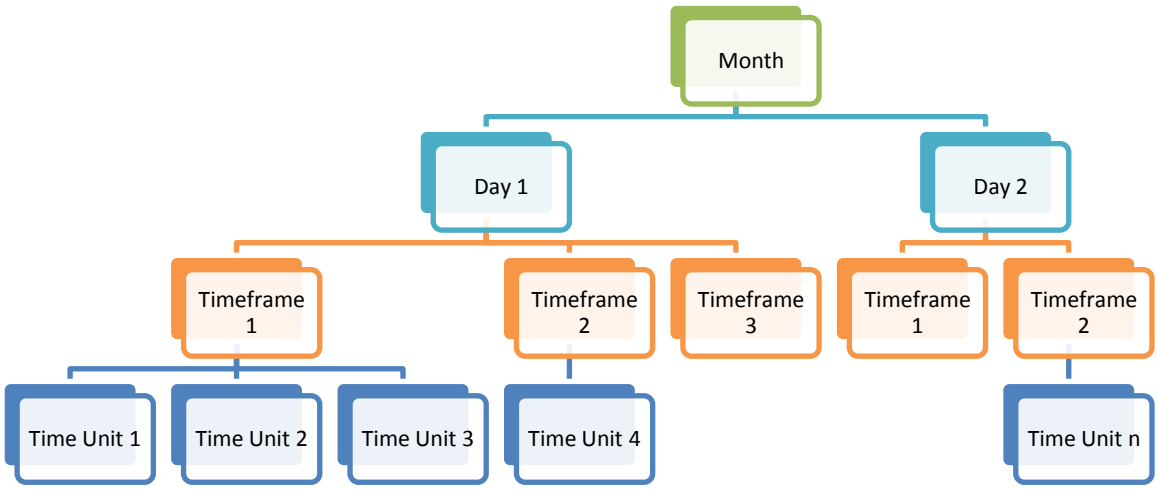

Figure 6 - Tree model of user consumption

The usability of the structure lies in how the time units are handled on each level. If the client would have the previously mentioned day-night-electricity, the hours under different time frames would have different pricing. Once the user logs into the system and asks for the current billing situation, the system would fetch all the currently available readings and calculate the price, displaying it to the user. When using the previously mentioned method, only the necessary entries are stored and the temporary database can be kept to a bare minimum. This kind of

structure is usable for various lengths of timeframes and time units. The structure also supports computations that happen outside the batch processing and can be used later on in batch processing.

3.2 Example Case

This example describes a situation where the energy meter readings are read from the metering database on different moments of time and the results are saved to a temporary database for later use. The example happens at two sequential moments of time.

First, the client logs in to his personal service, which displays the current energy consumptions for every hour. As the user has logged in to the service, it makes a connection to the metering database and fetches the latest readings. These readings are processed and built into a tree-like format. As there are currently only three measurements and they are charged according to same tariffs, they are placed under the same time frame. Two of the meter readings are marked as OK, but for some reason the third hour is just an estimate and will be marked as such. The current invoice and energy consumptions are shown to the client and then saved to the temporary database. The situation is shown in figure 7a (red values are estimates, blue values are correct).

Few hours later, the client returns to the service and asks for the current situation. Now, instead of querying everything all again from the metering database, the previously fetched and calculated values are retrieved from the temporary database. As the third hour was marked as an estimate, the processing unit automatically asks from the metering database if an update has been received. In addition, the new values since last query are also retrieved from the metering database. As a result, the previously estimated value has changed to reliable status and one more hour has been added to the tree. This situation is shown in figure 7b.

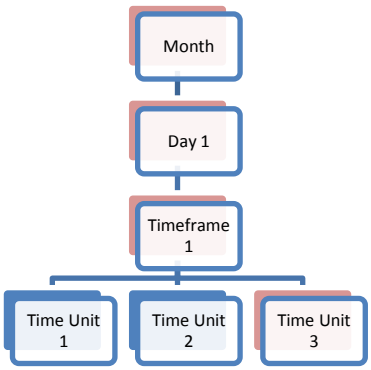

Figure 7a

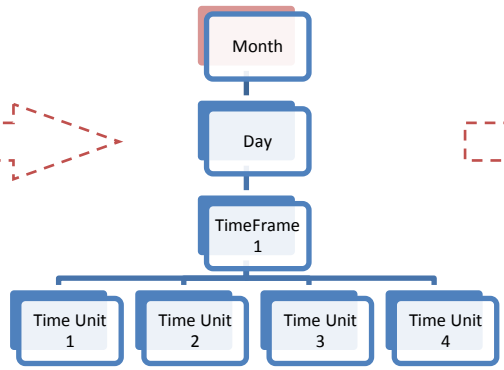

Figure 7b

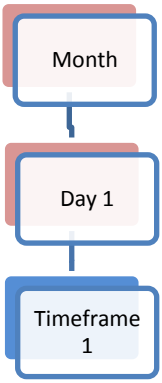

Figure 7c

As all of the values under the Time Frame 1 are marked as OK (blue), the metering tree can be pruned to save space and computation in the future. The time unit nodes are summed up and their total energy and money values are then saved to the time frame node. Once everything has been added to the time frame, all the nodes underneath can be removed, as there is no need for them anymore. Now the pruned tree can be stored in the temporary database and once the client asks for the invoice information again, there is no more need to calculate the hour nodes underneath – all their information is now stored in the time frame. The final situation is shown in figure 7c.

4 Experiments

This chapter describes the performed experiences. There are quite a many different aspects to consider in the given system, such as network transfer times, data sizes, computational complexity and so on. Here we evaluate the performance with a number of changing variables that are listed on Table 2.

Table 2 - Evaluated Parameters

Parameter	Values
Measurement interval	15 min, 60 min
Price intervals	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 parameter is the measuring interval, i.e. how often the meter readings are received. Two different situations were chosen as the basis – interval of once every 15 minutes and the other as an interval of once per hour. The change from one hour to 15 minutes means a noticeable increase to data size and should affect the computation.

The second parameter used is the price interval. This means on what different pricing intervals the invoice is created upon. Once per day means just a simple pricing, where the price of the metered data is always the same. Two intervals per day is considered to be a day-night electricity contract, where the customer has cheaper energy during night time and more expensive during day time. The last variation is 24 intervals per day, meaning that every metered hour will be taxed according to the real-time energy market prices.

The correct estimate percentage means the amount of meter readings that are correct. A 40% estimate means, that only 40% of received meter readings are correct and the rest (60%) are estimates or missing and have to be updated later.

The number of processor cores in the server is quite self-explanatory; these parameters point out the amount of processor cores are available for calculations. This information is used to compare the parallel processing to serial processing and how they affect the final invoice calculation times.

4.1 Test Hardware

The goal of the project was to consider different possibilities of running the calculation operations. For the sake of simplicity, an off-the-shelf computer was bought for testing purposes. The machine running the tests has an Intel Core i7-2600 quad-core processor that runs at clock speed of 3.40 Ghz. The computer is equipped with 8 Gb of RAM and is running a 64-bit Windows 7 operating system. The computer itself is not optimized for any of the possible solutions, but is something that can be easily purchased for server use.

On the other hand Microsoft offers the possibility of buying computation power from the Azure cloud service. For comparison purposes, the different virtual machines available are presented in Table 3 and Table 4.

Table 3 - Azure computation comparison

Virtual Machine Size	CPU Cores	Memory	Disk Space for Local Storage Resources in Web and Worker Roles	Disk Space for Local Storage Resources in a VM Role	Allocated Bandwidth (Mbps)
ExtraSmall	Shared	768 MB	19,480 MB (6,144 MB is reserved for system files)	20 GB	5
Small	1	1.75 GB	229,400 MB (6,144 MB is reserved for system files)	165 GB	100
Medium	2	3.5 GB	500,760 MB (6,144 MB is reserved for system files)	340 GB	200
Large	4	7 GB	1,023,000 MB (6,144 MB is reserved for system files)	850 GB	400
ExtraLarge	8	14 GB	2,087,960 MB (6,144 MB is reserved for system files)	1890 GB	800

Table 4 - Azure pricing comparison

Virtual Machine Size	CPU Cores	Memory	Cost Per Hour	Cost Per Month
Extra Small	Shared	768 MB	$0,04	21,43 €
Small	1	1.75 GB	$0,12	64,28 €
Medium	2	3.5 GB	$0,24	128,56 €
Large	4	7 GB	$0,48	257,12 €
Extra Large	8	14 GB	$0,96	514,24 €

*Note: Euro prices calculated with exhance rate of 1 USD = 0.743983 EUR

Even though Microsoft offers highly scalable computational capacity, it is somewhat overpriced. As Microsoft requires at least two instances to be running in order to receive the promised 99.95% service level, the price grows quite quickly. When compared with the off-the-shelf solution used for the testing purposes, the rent of large computation unit for few months already surpasses the price of purchasing own processing machine.

Table 5 - SQL database pricing in the cloud

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, $3.996 for each additional GB
Greater than 10 GB to 50 GB	$45.954 for first 10 GB, $1.998 for each additional GB
Great than 50 GB to 150 GB	$125.874 for first 50 GB, $0.999 for each additional G

Table 6 - Data transfer pricing

Pricing details for outbound data transfers
North America and Europe regions: $0.12
Asia Pacific Region: $0.19

4.2 Data size

Due to the nature of how databases work, the real data level information underneath the relational database is not easily calculated, but a decent level of approximation can be measured. In addition, the size comparisons of the tables are related to data transfer as well; that is the amount of data that needs to be passed from one service to another. On a byte level, the measurement tree contains the following entries:

Table 7 - Client entry breakdown

Parameter	Parameter Type	Example	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		

4.3 Transmission cost

The transmission cost for the invoice calculations are calculated according to the data size presented. The amount of 66 bytes of data transfer for each entry calculation does not appear to be significant in the beginning. However, the data size grows surprisingly fast and has a significant impact on the performance. The interval by which the metering is done creates an accumulative hindrance to the invoicing service. With 15 minute metering interval, the data amount grows up to 190 kb per user per month which is 400% of the data size of 60 minute intervals as shown in Table 8.

Table 8 – Data size in kilobytes per user

	Entries / day	Entries / month	Data / month
15 mins	96	2880	190,08
60 mins	24	720	47,52

Although the 190 kb/user data transfer does not look problematic it truly is. Once the invoice calculations have to be done for 1 million clients, the effects start adding up. As shown in Table 9, the data amount grows up to be 190 Gb for million customers – an amount that indeed is significant.

Table 9 – Data size in gigabytes per month

	300 000	500 000	1 000 000
15 mins	57,024	95,04	190,08
60 mins	14,256	23,76	47,52

When considering the data transfer amount and the fact that the case presented in this paper is just a rough estimate on all the data required to calculate the actual invoice, it can be stated that the data should not be transferred over long distances. Best solution would be to perform the calculations in the same location where the databases are located. This way, the data transfer should not be as limiting bottle neck as it otherwise could be.

4.4 Computations

One important part to consider is the proposition of the tree-algorithm. The tree can be considered to be a theoretically efficient solution, but it does have downsides. To understand better the impact that tree has to processing complexity, it is important to consider a few examples.

An inefficient situation would be something, where the tree would have a lot of time frames full of estimated values that prevent it from being pruned. In the worst case scenario the tree would have 1 full month, 30 full day nodes that each have 24 time frames underneath (each hour is separately taxed entry) and they would have 4 nodes of 15 minutes metering data. Other bad case to consider would be the situation of having 2 time frames with full hours underneath them for every single day. A comparison of few different combinations is shown in Table 10.

Table 10 - Comparison of tree complexity scenarios

	Months	Days	Timeframes	Time Units	Total Entries	Percentage
Full Quarters	1	30	24	4	3631	
Quarters - 15 Days	1	15	24	4	1816	50,01 %
Quarters - 15 days & ½ time frames	1	15	12	4	916	25,23 %
Full Hours	1	30	2	12	811	22,34 %
Hours - 15 Days	1	15	2	12	406	11,18 %
Hours - 15 Days & ½ timeframes	1	15	2	6	226	6,22 %

4.4.1 Calculation Costs

In addition, each of the calculations has a certain complexity to them. This is formed from the multiplications and additions that have to be done to the data in process. Each of the hours have a money and value parameters, that need to be multiplied together to get the energy price for the time unit. In addition, these hours have to be summed up in order to have the final monthly invoice.

The following tables portray the costs of data calculations. The cost of calculating one node is approximated on an operational level. The table of the metering tree contains 7 different values that need to compared and calculated. Each of these values needs to be processed in order to form the final invoice for the client. If all the values are in place, in correct format and all the calculations are done in a big bunch, that would require 30 days * 24 hours of information = 720

operations on the entries + another 720 to sum them up. The equation for calculating the amount of operations per entry would then be:

*(Days * Hours) * Operations/entry + Days * Hours = Total operations*

Making the total required operations to be 5760 for each client. The operations required for different amount of clients can be seen in Table 11.

Table 11 - Operations per client

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

4.5 Computation time

The computation times have been measured with the parameters listed above. In the first part of the tests, a locally running system was created. This system is used to evaluate the performance on pure computational level; thus all the networking was removed from this phase. The computer running these tests is a basic off-the-shelf Intel i7 quad-core computer, as mentioned before in the chapter 4.1 Test Hardware.

4.5.1 Total Runtimes

The runtime tests measure the effect of different time frames, measuring intervals and the amount of correct estimates. Only one core of the processor is used in order to simplify the results. The 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 search speed of the underlying database from the equation. The database values were generated for the first seven days, which the processing unit then retrieves and processes. The information is handled and stored in the tree format, as explained in chapter

3 Computation optimization.

After processing the seven day batch, another seven days were added to the database and this process was performed four times in total, thus simulating calculations that are done once a week to update the invoice status. All the estimated metering statuses were changed to correct at the generation of the next week and 50% of those reading values were changed in order to add more realistic updating and recalculation to the system.

All of the tests were run five times and then an average of the runs was taken. The results of the runtime tests can be seen in Figure 8 and in Table 12 and Table 13.

A quick look at the graph shows what was expected – the amount of time frames and the measuring interval both add up the complexity. However, the more correct the meter readings were, the less time it took to compute the invoice. For example, the 24 Timeframe calculations with 40% of correct readings take 2.5 seconds with hour metering intervals. With metering interval of every 15 minutes, the same calculation requirement rises up to 10 seconds! However, once the reading reliability closes to 100% the calculation times are reduced with surprising amount, dropping the required processing time to a mere 1 second, which is almost the same as the one hour measurement interval requires.

Based on these tests, there is no doubt that the meter reliability plays an important role. Querying for updates from the metering database takes time and is the single most important factor in the calculations. In addition, it has to be noted, that the computing times also increase with the measuring intervals – which is only logical, as there are four times more values to be processed.

Table 12 - 60 min runtimes

	40 %	60 %	80 %	100 %
1 TF - 60 MIN	1,5782000	1,0620000	0,6118000	0,2624262
2 TF - 60 MIN	1,3084000	0,9018000	0,8596000	0,3220322
24 TF - 60 MIN	2,5676000	2,1696000	1,7508000	0,9956996

Table 13 - 15 min runtimes

	40 %	60 %	80 %	100 %
1 TF - 15 MIN	8,21302122	6,87348728	5,5730798	0,3242324
2 TF - 15 MIN	8,3762000	6,9540000	5,5020000	0,4000400
24 TF - 15 MIN	10,0210000	7,9490000	5,0976000	1,0703070

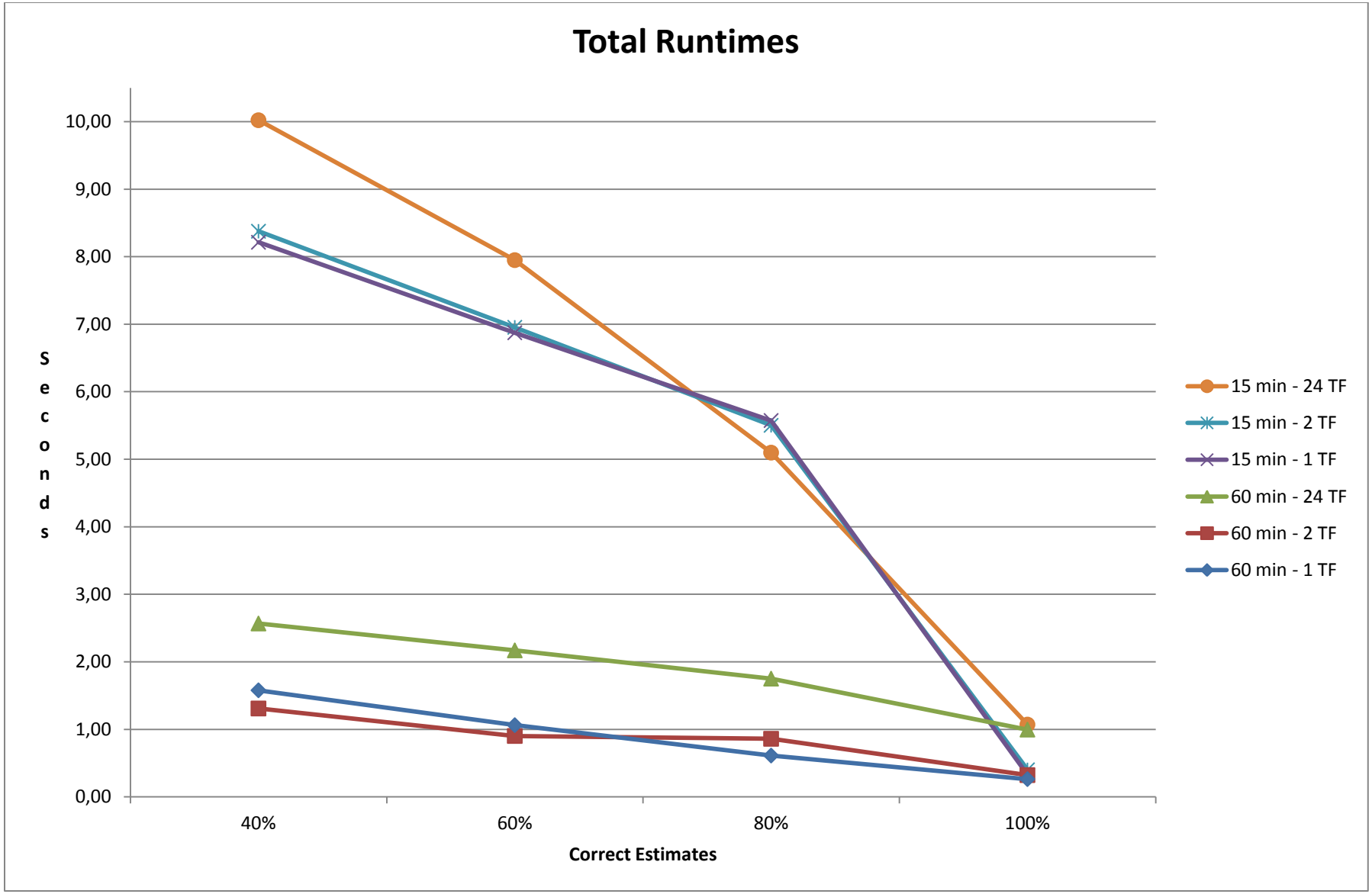

Figure 8 - Total runtimes of timeframes

4.5.2 Batch vs. Pieces

This test is about testing the difference between calculating the customer invoice at the end of month and calculating the invoice at certain moments of time. This test follows the procedures of the previous total runtime testing, but compares the "round times" of the runs to the results calculated at the end of the 28 days. In this test the times required to retrieve, update and calculate the current situation after each 7 day period are taken with the same changing variables as in the previous one. Each of the processing times of the 7 day period is calculated and the average of the runs is taken. This value is compared to the time it takes to calculate the final invoice at the end of the month. The results of these tests can be seen in Figure 9 and in Table 14,

Table 15 and Table 16.

Table 14 - 60 min round times (seconds)

	40 %	60 %	80 %	100 %
1 TF - 60 MIN	0,39455	0,2655	0,15295	0,065607
2 TF - 60 MIN	0,3271	0,22545	0,2149	0,080508
24 TF - 60 MIN	0,6419	0,5424	0,4377	0,248925

Table 15 - 15 min round times (seconds)

	40 %	60 %	80 %	100 %
1 TF - 15 MIN	2,05325531	1,71837182	1,39327	0,0810581
2 TF - 15 MIN	2,09405	1,7385	1,3755	0,10001
24 TF - 15 MIN	2,50525	1,98725	1,2744	0,267577

Table 16 - Batch runtimes

	60 MIN	15 MIN
1 TF	0,070152	0,130231
2 TF	0,128203	0,188055
24 TF	0,7726692	0,833071

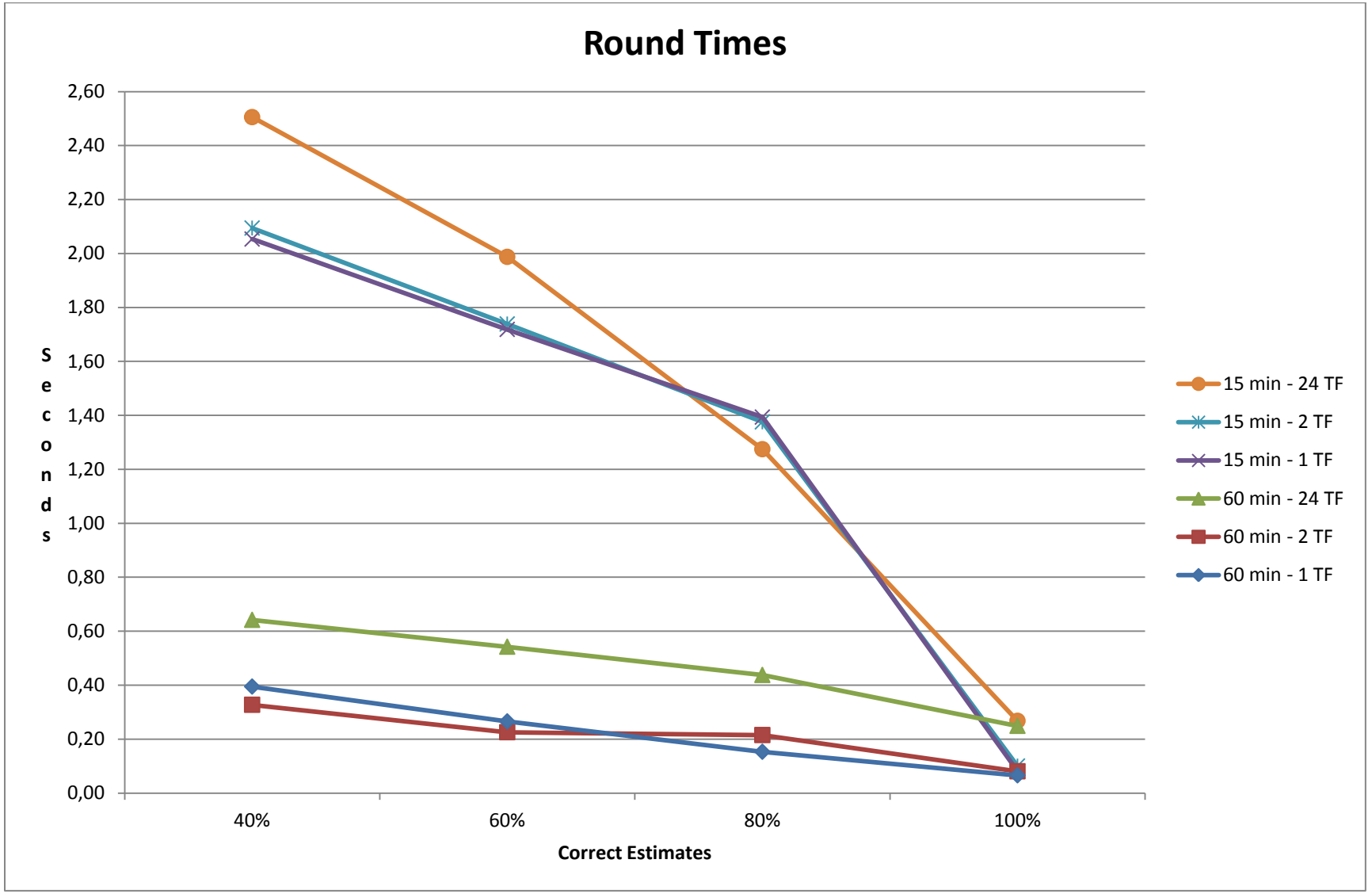

Figure 9- Round times

The results seem to point out that batch processing is quite efficient when there is unreliability in the meter readings. However, once the complexity rises to the 24 time frames per day, the batch processing becomes slower than running one round with the tree processing. In addition, when looking at the 15 minute interval measurements, it is clear that the estimate percentage adds to the processing speed. As in the previous tests, the slower processing is explained by the need to make queries to the metering database in order to update the estimated readings.

However, another important fact is the speed difference when all the values are always correct. Even with quite complex combinations (15 minute measurement intervals, 24 time frames per day), the weekly calculations are faster than running the batch at the end of the month. This is due the nature of the tree structure and how the pruning decreases the amount processing required.

Even though one 7-day update round can be faster than calculating the complete batch at the end of the month, the total time is still higher. Even in the best case where single calculation takes 0.24 seconds on average, calculating the batch at the end of the month takes 0.16 seconds less (as all the values are final and no estimates should remain at the end of billing period). The results do however point out that calculating and storing invoice data in sensible manner can ease the processor loads at end of month and provide lower response times on client queries.

4.6 Parallel Scenarios

The distributed scenario takes a look at solving the problem by utilizing threads to gain more efficient control of the invoice calculation process. In this scenario, the calculation task is split into smaller parts that are then handed out to different processor cores for processing. Each of the tasks is handled individually by the designated processor core. When done in this manner, the full processing power of the computer can be manifested for use.

In Microsoft .NET environment this can be done with just few modifications to the program code. The process has to be partitioned to tasks that are individual from each other and don't require much communication between each other. One solution in the .NET environment is to create all the tasks and place them into a task pool. The platform itself has automated the task switching quite well and can automatically pick tasks from the task pool. Once the task has been taken from the pool, it is run in parallel with other tasks and processed according to the process definition inside the task. Figure 10 demonstrates this process.

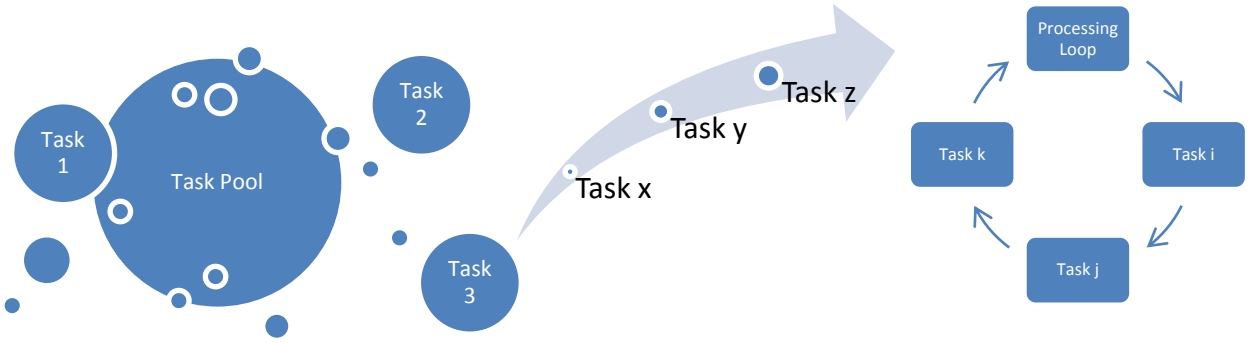

Figure 10 - Parallel task processing

In this solution, the user invoice calculations are instantiated as tasks. A task is the complete process that should be done to a client (fetch information from database, build the metering tree, prune the tree and finally save the results). As the task involves querying database and processing the information, a bigger batch of clients should be retrieved by each thread at once. This is to prevent the database of becoming a bottleneck, as all the customer data is not being queried all the time after each single result. Thus, the threading plan proposed fetches a certain number of users per task (e.g. 1 000) and then handles them all at once. As the threads will stop their processing at different times, the database is also queried at different moments of time, making it less likely to be queried by all of the threads at once. The whole task breakage is shown in the Figure 11.

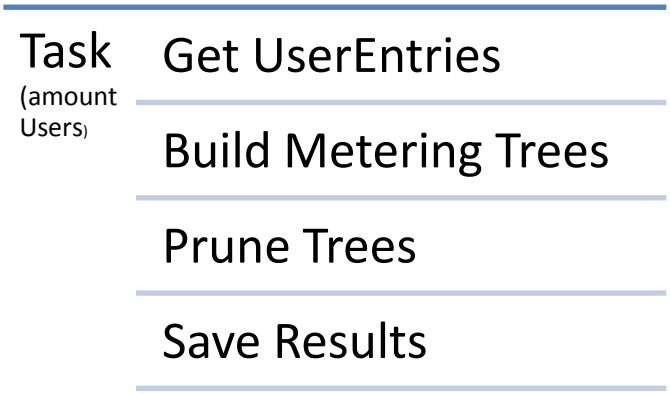

Figure 11 - Task breakage

The most important measurement to carry out is the difference between parallel and serial processing. The computation was performed as end of the month batch, where everything has to be calculated for each of the users from scratch. In addition a comparison was made against the solution of calculating the invoices at certain moments of time. The comparison simulates a situation where all the values are updated once every week and calculates only the missing week at the end of the month.

The tests in this case were run against a database of 1000 clients. The process was split into 125 tasks, leaving each of the tasks to contain 8 users with each user having 672 entries. The tests were run 100 times resulting in an estimate of total computation times of 100 000 customers. The comparisons were done with 3 different processing possibilities; Serial batch processing (1 core), Parallel batch processing with 4 cores and Parallel end of the month processing with 4 cores. The results of the tests can be seen in Table 17, Table 18 and Figure 12.

Table 17 - Execution times in hours

Clients	1 Core Batch	4 Cores Batch	4 Cores - Final Week
100 000	5,66	2,91	1,15
1 000 000	56,57	29,06	11,51

Table 18 - Comparison of execution solutions

	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 %

As can be seen from the results, the parallel processing truly enhances performance of the invoice calculation. Even though the calculations are run on a off-the-shelf computer, the parallelism brings a big boost to the performance. Even with one million clients the execution time can be taken down to 29 hours, which is almost half of the original execution time. When adding the possibility of performing calculations in smaller pieces, the updating of entries at the end of the month drops down to under 12 hours. That is only 20% of the original time requirement.

However, it has to be noted that the task size (how many users per task) and the growing size of the database add to the problem. The amount of tasks that can be held in the memory while processed is related to the amount of memory available. In addition, the database underneath can have some performance teardowns as the size grows bigger. One thing to consider as well is the amount of threads to run in a machine. A good rule of thumb is to run threads twice the amount of cores, but this might not be always the case.

Based on the experiments carried out, it can be stated that parallel computation brings a notable enhancement to the invoice calculation. When combined with partial processing the parallel execution

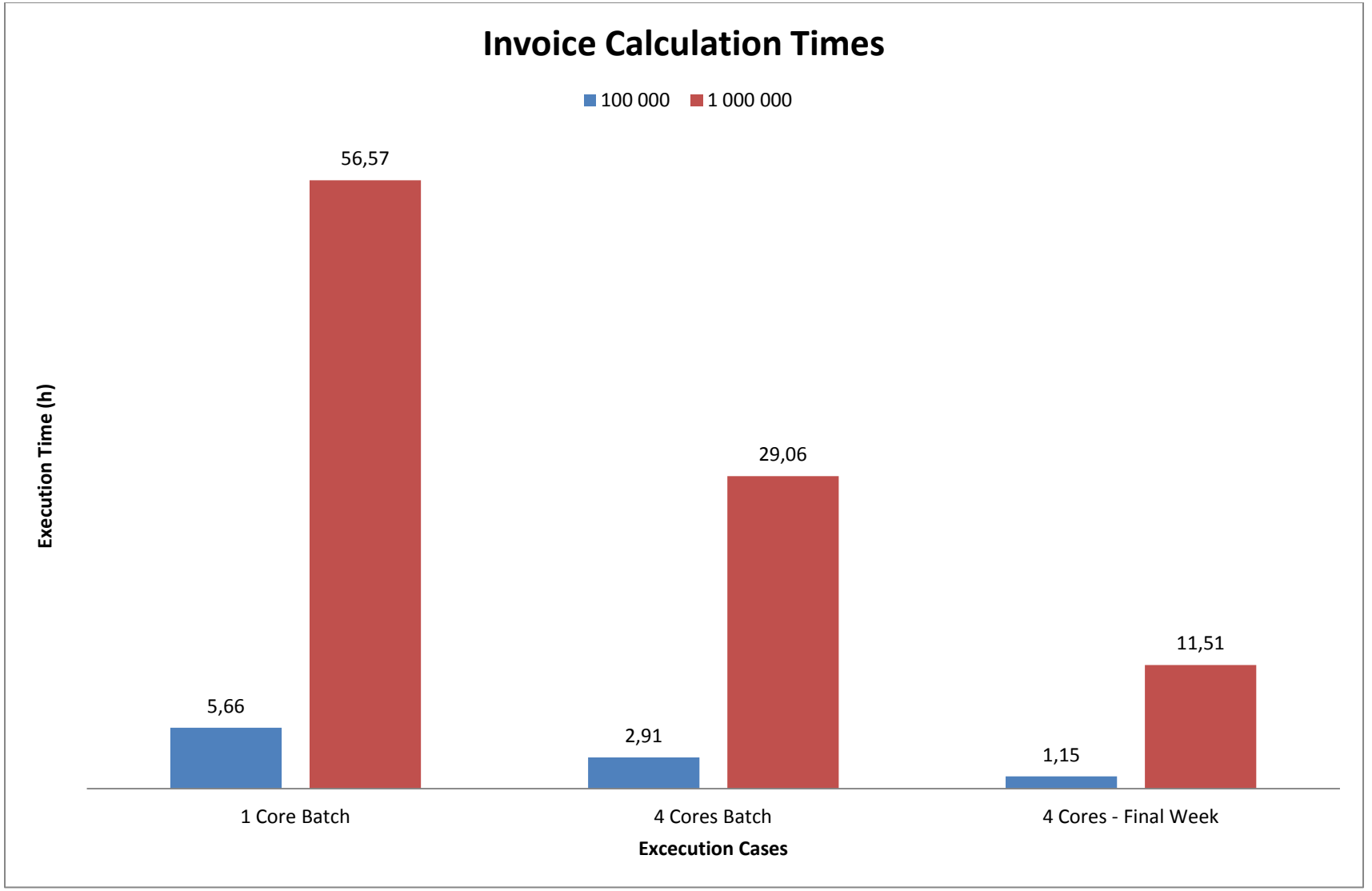

Figure 12 - Invoice calculation times

5 Conclusions

The results attained within this project seem to have a certain trend. As the data size grows to huge proportion with the increase of metering interval frequency, the processing requirements grow as well.

Calculating the clients' invoices all at once is one option that does not require huge amounts of development time. However, as seen in Table 17 and in Figure 11 the calculation time for million clients would be over 56 hours at the end of the month. Instead of doing batch processing in a serial manner, partitioning the invoice calculation to smaller tasks and calculated in parallel with multiple processor cores brings a noteworthy change in the processing time. The parallel processing with a quad-core computer drops the calculation time down to 50% of the original to a little over a day (Table 18).

Another solution to the problem would be to calculate the user invoices during the measurement period, e.g. once every week, and save the results. This approach would lessen the computational burden at the end of the month as parts of the invoice have already been calculated. The optimization algorithm shown in chapter 3 introduced a tree-form solution that remembers previous calculations and only updates dates that have changed. It already requires only a little amount of storage space, is fast to process and requires less data transfer when updating the previously calculated values. When the tree algorithm is run in parallel utilizing all the available process cores in the test machine, the calculation time of 1 million clients drops down to 11 hours. The result means a change down to 20% of the starting situation of calculating all the invoices as a serial batch (Table 18, Figure 12).

When looking at the data size that should be transferred, the usage of cloud does not look quite reasonable. With 1 million clients, it would require 190 Gb of data transfer each month to calculate all the clients invoices. However, if the data would be stored already in the cloud the transfer problem would disappear, as the data would reside within the same premises as the processing unit. The problem with cloud computing comes from the high pricing of processing time. Even though the storage space is rather cheap, the processing power is not. Renting cloud computation that is equivalent to the used test computer, the price of the machine would be spent in few months, as can be seen from the Table 9. Considering the requirement from Microsoft's side to purchase at least two processing units in order to gain the promised 99.95% uptime, the price does skyrocket rather quickly. When looking at the problem from the monetary side, purchasing and maintaining own processing unit is the better option as stated in chapter 4.1 Test Hardware.

The event-based invoice calculation, where the user invoice is calculated upon the client's request is possible to implement even with regular off-the-shelf computer, as described in chapter 4.5 Computation time. The Figure 9 shows that calculating invoice data of one week does not take more than 2.5 seconds even in the worst-case situation, where metering data is

measured every 15 minutes and most of the meter readings are estimates. Storing the results received by such on-demand calculations is suggested, as this reduces the final end of the month calculation dramatically (Table 14,

Table 15 and Table 16). Thus the result suggest, that calculating user invoices on-demand is a viable solution that not only performs fast, but saves the time required for the final end of the month computation as well.

Considering all the facts mentioned above, the best solution would be to implement the service as a local, on the premises solution. The cost for purchasing own servers outmatch the costs of the Microsoft Azure cloud. Parallel processing provides a significant gain to processing speed and the service should definitely be developed using parallel approach. In addition, if the clients are given the possibility of monitoring their own energy consumption and invoicing situation over the internet, it would be useful to save the already made calculations for later use. As stated before, calculating the client invoices at certain moments of time, e.g. once every week, makes the final invoice creation rather fast.

6 Future Development

Considering the information security is a vital part of the invoicing service. Lappeenranta University of Technology is also doing research in work package 4.1.2 Trust and Privacy, where the security of individual client is considered. The next natural step in the development process of invoicing service would be to combine both the results of this research and the one done in Trust and Privacy.

Another part to consider is the development of the service with a SOA (Service Oriented Architecture) mindset. The ideology behind this approach is to build the system to support the real business logic underneath. As all the business units can be decoupled from each other, their communication interfaces can be abstracted. Implementing the service in such a manner would allow the system to function more efficiently in the long run, allowing modifications in the business logic, but preserving the already existing interfaces. Thus, the system would not impose any changes to other services that take advantage of the offered services.