CHALMERS

BCStream - a data streaming based system for processing energy consumption data and integrating with social media

Master of Science Thesis in the Programme Computer Systems and Networks

CARL WÅLINDER
BAO HOANG

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, February 2015

BCStream - a data streaming based system for processing energy consumption data and integrating with social media

Carl Wålinder
Bao Hoang

Examiner: Landsiedel, Olaf

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden February 2015

Abstract

Lowering energy consumption is one of the biggest challenges in today's society. Each and everyday we use a large amount of electricity to support our lifestyle. According to the U.S. Energy Information Administration, the energy usage will increase by 56% by 2040. Thus new innovative solutions to reduce power consumption are in great need. Consumers typically do not have the information they need to take proactive decisions about their energy usage. Therefore, the potential for saving energy might rely on the awareness and involvement in energy conservation and waste reduction.

One of smart grid's visions is to enable utilities and consumers to more effectively monitor, control energy usage and cost. Advanced Metering Infrastructure (AMI) is an important part of smart grid, it is a two-way communication system between smart meters and utilities, where smart meters transmit consumption data to the utilities. Smart meter is a digital metering device, which not only shows real-time feedback of the energy consumption but also supports remote communication with utilities. Consumers can receive their power consumption data in real-time, which allows them to make informed choices about their energy usage. In this sense, smart grids, AMIs and smart meters can be used to make people aware of their energy consumption. Additionally, energy usage trends can be illustrated to consumers by applying different queries to extract energy consumption data from the AMIs. At the same time, social media is one of the biggest communication tools used to reach millions of people. By integrating energy usage data produced by AMIs on social media, consumers can inspire and influence others to match the energy they use with their needs and lifestyles.

As more and more smart meters are installed across power grids, huge volumes of data are generated. In order to capture and analyze large data sets generated by smart meters within AMIs, challenges such as how to cope with huge volumes of data appears. Data Stream Processing models, utilized to process the huge amount of continuous data and extract interesting data, emerge as a possible approach to transform these real-time operational data into applied insights.

This master thesis presents BCStream, a data streaming based application, which consumes energy consumption data on the fly and produces feedback to users on social media in order to raise their energy usage awareness. BCStream utilizes a Stream Processing Engine to process a large amount of continuous data in real-time. In order to show that it is possible to address the aforementioned challenges, we implement and evaluate BCStream using several use-cases. These use-cases provide consumers different insights of their energy usage. For instance, in the first use-case, the proposed system determines at which hour of a day and which day of a week the power is consumed the most. In the second use-case, BCStream helps users find the top five houses using the most energy in the consumers' neighborhood. Finally, the third use-case locates top five most energy consuming devices per household. In order to evaluate the performance of BCStream, real energy readings are modified and used as input data to the use-cases. In order to conduct the evaluation, we use a single machine that relies on a multi-core architecture to demonstrate good performance (between 160,000 and 170,000 energy consumption readings) and scalability.

Keywords: Energy Saving, Data Streaming, Stream Processing Engine, Smart Meters, Social Awareness, Smart Grid, Storm.

Acknowledgements

We would like to thank our supervisors Vincenzo Gulisano and Giorgos Georgiadis and examiner Olaf Landsiedel for their advice and guidance during this master thesis.

Carl & Bao, Gothenburg 2015/02/03

Acronyms

AMI	Advanced Metering Infrastructure
CEP	Complex Event Processing
CSV	Comma-separated value
DAG	Directed Acyclic Graph
DBMS	Database Management System
DSM	Demand Side Management
DSMS	Data Streaming Management System
DSS	Demand-Side Management
EIA	Energy Information Administration
FIFO	First In First Out
GC	Garbage Collection
MDMS	Meter Data Management System
SNS	Social Network Sites
SPE	Stream Processing Engine

Contents

1

Introduction

A more sustainable living plays a key goal in our modern society where the number of electrical devices in our homes is growing. In order to lower the energy usage, the first step is to make people aware of how much energy they are using and relate it to their daily behaviours. The main current approach for consumers to get feedback on their energy usage is through their monthly bills. By receiving feedback more often, the consumers' attention in energy usage could be raised significantly [1, 2]. A lot of effort has been put into this area, where integration between consumers and suppliers is done with continuous exchange of consumption data using Advanced Metering Infrastructures (AMIs). Moreover, integrating energy consumption data into modern social media enables users to share their achievements in energy saving with friends and family. They might receive even more attention and encouragement to further reduce their energy footprint. Widely used social media such as Facebook and Twitter can be one of the best ways as information sharing tools which can be utilized to raise energy usage awareness among their users [3].

This section first raises the problem and challenges to capture and analyze the huge amount of energy consumption data generated by AMIs and provide useful insights to consumers. Then it continues with the project's goal, contributions and finally a short outline of the report.

1.1 Problem description & challenge

Would you run the dishwasher or washing machine in an off-peak time if you know how much less expensive the energy is at that time? Would you do it if you receive a notification ? These are key questions for not only the energy utilities but also for the consumers. As mentioned above, one of the key problems with improving the saving of energy is getting consumers to be aware of their habits and choose to do something about it. Until recently, there has been a lack of information to cause such a change. However, the deployment of smart meters, digital meters used to measure energy consumption, in AMIs enables the ability of tracking and reading energy consumption data within the power grid. We are now able to perform detailed analytics of energy usage data to provide consumers valuable insights to change their behaviors and costs. Smart meters provide many new opportunities, but also new challenges. One major challenge is the massive amount of data that can be created by the smart meters. For example, if a distribution network has one million metering devices and 5 kilobyte record per collection the amount of data can grow exponentially quickly. Table 1.1 shows the potential amount of data generated by one million metering devices in a year:

Collection Frequency	1/day	1/hour	1/30 min	1/15 min
Numbers of collected records	365 M	8.75 G	17.52 G	35.04 G
Terabytes of collected data	1.82	730	1460	2920

Table 1.1: The table indicates the potential amount of data generated by one million smart meters.

Given the volume of data being generated and the ability to provide energy consumers different insights of their energy usage on the fly, the main challenge of this master thesis is the complex analysis of the input energy data in a real-time fashion.

1.2 Goal

Research has shown that monitoring energy usage contributes to energy saving (see Section 6.3). By providing energy consumption information on social media, this thesis proposes a way to utilize the influence and power of social media to reach out to millions of people in order to get their involvement in energy saving. The main goal of this master thesis is to present BCStream, a data streaming based application, which is able to handle big and unstructured data streams, generated by smart meters, in near real-time. Furthermore, it continuously performs data analysis to capture the most interesting insights to consumers' energy usage from the input stream despite the size of the stream. Consumers are able to specify what kind of feedback they are expecting from BCStream and how it is presented, e.g., by e-mail, text messages or on social media. BCStream makes it easier for consumers to view and evaluate their energy consumption and consequently consumers can raise their energy usage awareness significantly, which might result in more informed and proactive actions to save energy.

Following questions are investigated and evaluated:

- How can BCStream extract useful insights to consumers' energy usage from unstructured data with various format from different devices in a continuous fashion?

- Can BCStream provide its users the ability to construct their own queries ? If this is the case then how?
- How would BCStream perform focusing on its throughput (energy readings/s) and latency (ms). How would the system performance be effected if more hardware, e.g., increasing the number of processor-core, is utilized?

1.3 Contribution

The main contribution of this thesis work is the design and implementation of BCStream. Using BCStream, a great number of use-cases can be composed to provide consumers different insights. Furthermore, we conduct a questionnaire to investigate potential users' interest in using BCStream for saving energy. The result indicates that people are very positive to receive insights to their energy usage. We also implement and evaluate sample use-cases with extensive experiments to measure BCSteam's performance focusing on throughput (energy readings/s) and latency (ms). Real energy consumption data sets, obtained from TraceBase [4], are modified and used as input data stream. The evaluation shows that, by using an off-the-shelf server, the system can handle up to 170,000 energy readings per second and latency as low as 14 ms. Moreover the system achieves better throughput with increasing computing power in terms of number of CPU cores.

1.4 Outline

The report continues with the following Chapters:

- **Chapter 2: Background** This chapter introduces all concepts needed to understand the rest of the report including social media, smart grids, AMIs, data streaming and stream processing engines.
- **Chapter 3: System design and architecture** This chapter introduces BCStream's architecture and an overview of the modules that compose it.
- **Chapter 4: Use-cases & implementation** This chapter represents the use-cases implemented in BCStream and how they are constructed using standard operators.
- **Chapter 5: Evaluation** This chapter represents the evaluation of the system including the different performed experiments.
- **Chapter 6: Related work** This chapter brings up other studies related to this master thesis.
- **Chapter 7: Discussion** This chapter provides the conclusion of this master thesis and what could be done in the future in terms of further development of BCStream.

2

Background

In this chapter, we introduce the necessary background in order to understand the basic concepts for the report. It appears to be an approach to get people involved in energy saving by providing energy consumption insights on social media [3, 5]. Section 2.1 introduces social network sites and a discussion on how social media can be used to influence a group of people. At the same time, in order to provide consumers with useful insights of their power consumption, energy readings obtained from smart grids and Advanced Metering Infrastructures (AMIs) are needed. Section 2.2 introduces the components and concepts of smart grids and AMIs. Finally, Data Stream Processing model, utilized to process huge amount of continuous input streams and extract interesting data, emerges as a possible approach to transform these energy readings data into applied insights. In Section 2.3, we present Data Streaming paradigm.

2.1 Social media

When something is unique or creative it gets attention, information that we think others like are shared. In the past, this sharing was done via word of mouth, publications or on TV. Now it is done online. Social Media is the new way to share information, projects or images we like with the world. People are spending more and more time online, and a large part of that time is spent on Social Network Sites (SNSs), which are introduced in the next Section 2.1.1.The increasing number of SNSs users makes social media become a powerful tool for exchanging information. Consequently, the popularity of social media enables opportunity to gain influence and engagement among the audience. Section 2.1.2 presents social influence phenomenon.

2.1.1 Social Network Sites

Facebook, Youtube and Myspace are all examples of SNSs. SNSs are defined as online applications where users can share activities and interact with others. Boyd et al [6] present three properties that define a SNS: (1) the ability to create a profile which is visible for others; (2) to have a list of connected users; (3) to be able to explore other users' connections. The term network is not necessarily related to “networking”, which emphasizes on finding new relations to strangers. Even though finding new relations through SNSs occurs, the intended use is to maintain contact with family, friends, colleagues, etc.

2.1.2 Social influence

Social influence is the phenomenon where actions, performed by humans, affect others to act in a similar way, this could be friends or complete strangers. These actions do not necessarily have to be physical actions, it can in fact be small acts like re-posting someone's comment or “liking” a status update. Social influence is a rather explored area and interesting results regarding how one can be influenced by others' actions on social media have been presented in [7, 8, 9]. Goyal et al [8] investigate whether it is possible to model the influences by looking at social graphs and logged actions. In their result section, they state “we are able to predict, to within tight margins, the time by which an influenced user performs an action after its neighbors have performed the same action”. Example of such an action could be that a user joins a group on a social network and then it potentially could intrigue other connected users to join the same group.

With this knowledge, it is positive that people get affected and competitive with each other on social media. Since our objective is to influence people to lower their energy consumption and also to become more aware of their own energy usage, it is clear that the use of social media, with its effect on its users, is a great way to actualise this. However, consumers typically do not have the information they need to make informed, proactive decisions about their energy usage since the current approach for consumers to get feedback on their energy usage is through their monthly bills. In order to provide consumers such feedback, energy readings obtained from the smart meters within smart grids and AMIs are needed. In that sense, we introduce smart grid, AMIs and smart meters in the next section.

2.2 Smart Grid

The current electric grid was built over a hundred years ago. Back then, the electricity cost was low and the result of that was a system not concerning about conserving surplus energy [10]. However, with the increasing energy usage, the electric grid is reaching its maximum capacity. To move forward, a new kind of electric grid, the smart grid, is built to utilize new smart appliances such as smart TVs, dishwashers, etc., in order to manage the increasing complexity and needs of electricity in our modern society. At the same time, one of the key problems with improving the energy saving is engaging consumers to be more aware of their energy usage. Until recently, there has been a lack of information to cause such a change. However, energy consumption data becomes available using

AMIs and smart meters within smart grid. Such data is analyzed to provide consumers with useful insights to make them aware of their energy usage. This section gives a brief introduction to smart grids, followed by AMIs and smart meters.

2.2.1 What is smart grid ?

A smart grid is an electrical grid that extensively uses communication for improving energy efficiency and keeping track of the supply and demand within the grid. This communication makes it possible for automated adjustment and control of millions of devices [11]. Smart grid is not just about utilities having control over the electrical network and constantly knowing the demand and supply the minimum voltage needed, it is also a way of giving consumers more knowledge and feedback about their consumption, this is referred to as *smart generation* [12]. Other components, included in the smart grid, are smart power meters and intelligent appliances.

Smart grids are intelligent systems, they administrate the distribution of energy along the grids and can be used to optimize the use of renewable energy. Furthermore, the two-way communication between suppliers and consumers makes it possible to observe peaks in energy demand in real-time and to ensure that the demand is met. Smart grids can also help the reliability of the system and make sure that the customers would be provided with a stable power supply [13].

2.2.2 Advanced Metering infrastructure

AMI is a two-way communication system between meters and utilities [14] where the customer-end meters transmit consumption data to the utilities. It is a further development of the Meter Reading System, which was introduced to reduce cost and improve the accuracy of meter reading. As the name indicates, AMI is not just a single technology but rather a whole infrastructure with the possibilities of configuration and integration with existing electrical systems in our homes. The customer-end devices could be energy management systems or smart meters and these devices are all a part of the smart grid. Consumption data is not only sent to the utilities, it can also be displayed to the user for instant feedback in some cases.

2.2.3 Smart Meter

A smart meter is a digital metering device, and in this case it is about electricity metering. The word *smart* in smart meter refers to the fact that it not only shows real-time feedback of the actual consumption, it also supports remote communication and controlling for the suppliers. Furthermore a smart meter can directly connect to the electricity provider to constantly provide them with the consumption data. To give evidence of the widespread use of smart meters, the installation of smart meters are growing, in 2020 most of the households in UK would have one installed in their home according to [15]. As more and more smart meters are installed, this leads to huge volumes of data being generated. In order to capture and analyze large data sets generated by smart meters, Data Stream Processing models, utilized to process the huge amount of continuous data and extract interesting data, emerge as a possible approach to transform these real-time operational data into applied insights. The next section presents the concepts of data streaming.

2.3 Data streaming paradigm

There are a large number of sensors, smart meters, etc., being deployed across the power grid infrastructure and that results in huge amount of data which is accessible at unprecedented scale [16]. Consequently, more proactive decisions can now be made based on the available data. However, with traditional computing, data is collected, stored in a specific system, e.g., a database, and databases are first designed to efficiently store data, rather than rapidly process it and scaling accordingly. Therefore it is not suitable to use the traditional computing in order to capture and analyze real-time energy usage data. Stream computing is a new paradigm which supports processing data in motion and stream processing applications can be deployed to undergo complex analysis of continuous data streams in a real-time fashion [17]. This section first raises the importance of data streaming, and furthermore provides an overview of the stream processing paradigm including its key concepts.

2.3.1 Why is Data Streaming important?

As more and more intelligent devices such as sensors, smart meters proliferate across power grids, a large volume of data is available for instant processing as it is being produced. Real-time analysis of continuous data streams enables faster and more informed insights, which help consumers to better manage their energy consumption in order to take action in time. However, because of the data volume produced by all the devices across smart grids, it is not suitable to store all the received data into storage for later analysis as in the traditional computing. A data stream processing application can process continuous data streams on the fly, reducing large-volume input data streams to low-volume output data streams for further storage or computation. Aside from the above-mentioned motivations from the application perspective, stream processing is also motivated by current technology trends. Multi-core processors and clusters of servers are becoming more and more common. Thus data streams can be processed on different cores of a processor or on different servers of a cluster, which can enhance the stream processing application's performance as well as its scalability.

Stream processing is especially suitable for applications that exhibit the following application characteristics: intense computing (high ratio of operations to I/O), demanding low processing latency and ability to apply data pipelining where data is continuously fed from producers to consumers.

2.3.2 Data Streaming model

Data Stream

Applications such as smart metering based monitoring generate and push data to computation servers for real-time processing. The data generated by these applications can be seen as streams of tuples. A stream is an unbounded, append-only sequence of tuples generated continuously over time.

Tuple schema

Tuples contains one or more fields (F_1,F_2,F_3,...,F_n). For instance, a field *timestamp* could represent the timestamp when the tuple is created. Table 2.1 presents a sample tuple schema, generated by real energy consumption of electrical appliances provided by Tracebase [4].

Field Name	Field Type
Timestamp	*string*
Consumption	*double*
Device	*string*
CustomerID	*string*

Table 2.1: Sample tuple schema which contains four fields (Timestamp, Consumption, Device, CustomerID).

Fields *Timestamp* and *Consumption* represent the timestamp when the tuple is created and the wattage at that timestamp, respectively. Field *Device* determines the specified electronic device which the tuple is sent from and *CustomerID* represents the identification of the customer whom the device belongs to. For this project we assume that the field timestamp is always defined in the schema and that the tuples are sorted based on the value of it.

2.3.3 Query processing

A lot of architectures have been proposed to process data streams. Data Streaming Management Systems (DSMSs) process streams using continuous queries [18, 19]. These queries are translated into a plan of operators and can be optimized and executed. Queries over continuous data streams are very similar to traditional queries used in DBMSs. However, there is a major distinction between the two types. The traditional queries can be seen as one-time queries and the continuous data streams' queries as continuous queries [20]. The *one-time* queries are queries that are evaluated once over the data set with the answer returned to the user. *Continuous queries*, on the other hand, are evaluated continuously as data streams continue to arrive. The results to the continuous queries are produced over time and updated or stored as new tuples arrive.

Continuous queries

Data streams are consumed and produced by continuous queries. As in StreamCloud [21], a continuous query is defined as a directed acyclic graph (DAG) with additional input and output edges. Each vertex is an operator that consume tuples from one or multiple input streams and produces one or multiple output streams. An edge from operator A to operator B indicates that operator B consumes tuples produced by operator A. Queries are defined as continuous because they are executed repeatedly until they are explicitly removed and results are computed as new tuples arrive.

Data Streaming Operators

A stream operator takes one or more streams of tuples as input and outputs one or more streams of tuples. In most DSMSs, a stream operator behaves like its relational database counterpart. Data streaming operators are classified into two categories regarding whether they maintain any states while processing input tuples or not: stateless and stateful operators. BCStream defines the same streaming operator semantics as the ones in StreamCloud [21]. Each operator is defined as:

$$OP_N\{P_1,...,P_m\}(I_1,...,I_n,O_1,...,O_n)$$

OP_N represents the operator name, whereas, $\{P_1,...,P_m\}$ is a set of parameters, e.g., user-defined functions to transform input tuples or parameters and I,O denote the input and output stream.

Stateless operators

Stateless operators produce a result based on the evaluation of a single input tuple. Stateless operators do not maintain states that evolve accordingly with the input tuples being processed.

Typical stateless operators are map, filter, and union.

- **Map** - the data streaming counterpart of the relational *projection* function. The map operator accepts a single input tuple and produces a single output tuple with an arbitrary number of output tuple fields. It's main objective is to add, modify or drop fields from the input tuples.

  $$M\Big\{F_1 \leftarrow f_1(t_{in}) ,..., F_n \leftarrow f_n(t_{in}) \Big\}(I,O)$$

  Where I and O indicate the input and output streams. t_{in} is the input tuple and $\{F_1,...,F_n\}$ is the output stream's schema. The operator modifies each input tuple using user-defined functions $\{f_1,...,f_n\}$.

- **Filter** - the data streaming counterpart of the relational *select* function. The filter operator receives a single input tuple from the data stream and applies a user defined function to decide whether or not to keep that tuple. The *Filter* is defined as followed:

$$F\Big\{f(t_{in})\Big\}(I,O)$$

  I,O represent input and output streams, respectively. t_{in} is an input tuple and the operator decides whether the tuple is discarded or not using the user-defined function {f}.

- **Union** - the data streaming counterpart of the relational *union* clause. The union operator merges two or more input streams to a single output stream, all streams share the same schema. The operator is defined as:

$$U\{\}(I_1,...,I_n,O)$$

  Where I_1,...,I_n is a set of input streams and O is the single output stream

Stateful operators

Stateful operators process a set of input tuples, maintain the states and finally produce a single output tuple. In order to collect a set of input tuples, a window is defined and all tuples belonging to the same window are processed together. The window is a parameter of the stateful operators.

Windowing

When processing potentially unbounded sets of data it is impossible to execute functions such as average or sum, this is one of the motivations for using windowing [22].

Another aspect in the motivation is that the most recent data is considered, this is an important factor to some applications. For instance, a re-alarm application emphasizes data of the last thirty minutes than data from one month ago. Windowing is used to limit the scope of input data. The query is not evaluated over the entire data stream, but rather only over windows of recent data. A window continuously selects a part of the data stream, e.g., the last ten tuples or only tuples from the last hour and only considers these in producing answers with older data being discarded.

A window has a window start, window end, size, and advance. Size is the window's length and advance indicates how the window is updated. Whenever the window's start and end updates, the window updates and the tuples outside of the window are discarded. Depending on the relation of advance and the window's size, there are three different window types:

- **Sliding window** (advance < size): Sliding window (advance < size): The window start and end update depending on the being processed tuple's timestamp. When a sliding window updates, the window shifts to include the incoming tuple, and tuples still within the window are kept.
- **Tumbling window** (advance = size) is very similar to the sliding window. The main difference is that when a tumbling window updates, it shifts the current window by the window size units. Consequently, all the tuples within the new updated window are kept.
- **Jumping window** (advance > size) shifts the current window by the window advance units which can result in holes where data is being discarded. Consequently, all the tuples within the previous window expire.

Windows are differentiated into two different types; time-based and tuple-based window, e.g., time-based windows (also referred to as logical windows) are defined over a period of time (e.g., tuples received in the last 10 minutes) or tuple-based windows (also referred to as physical windows) are defined over the number of stored tuples (e.g., last 50 received tuples).

A typical stateful operator is the aggregate:

- **Aggregate** computes aggregate functions such as mean, count, sum, min, max over windows of tuples. The operator is defined as:

  $$A\Big\{ \mathit{Type},\mathit{Size},F_1 \leftarrow f_1(W)\ ,...,\ F_n \leftarrow f_n(W),\ [\mathit{group\text{-}by} = (F'_{i1},...,F'_{im})]\ \Big\}(I,O)$$

  Where I,O are input and output streams. Parameter *Type* specifies the window type which can be either time-based (*Type* = *time*) or tuple-based (*Type* = *tuple*). Tuples over input stream I are stored in the window W until it becomes full, which means that a tuple outside the window is received.

  The time-based window is considered full if a newly received tuple's field *Timestamp* exceeds the window *End* and the tuple-based window becomes full when the number of stored tuples is the same as the window *Size.* After an output tuple is propagated, the window is updated and the contributing tuples are discarded. (F_1 ,..., F_n) are output schema fields which are modified using user-defined functions (f_1, ... f_n). Group-by parameter is the data streaming counterpart of the relational *group-by* statement. The group-by parameter is optional and used to group input tuples based on the value of one or more fields such as (F'_{i1},...,F'_{im}). Whenever the group-by parameter is set the output tuple schema contains a timestamp and the input field utilized by the group-by. Finally, I and O are input and output streams.

2.3.4 Stream Processing Engines

Stream Processing Engines (SPEs) process continuous queries and produce results which are updated with the arrival of new data. Many researches have been undertaken and some well-known SPEs are Aurora [18] and StreamInsights [19]. SPEs can be implemented as centralized or distributed. Aurora is a centralized SPE, which has a single instance system executed in a single machine where all the processing takes place. The major issue with Aurora and other centralized SPEs is that the system capacity is limited to the capacity of the single machine and therefore its scalability with respect to the incoming stream volume cannot be achieved. Aurora is integrated into Medusa [18] to become one of the first distributed SPEs.

There are many types of distributed SPEs, where the most basic ones can run different queries on different nodes which makes it possible to scale the number of queries with more machines. More advanced distribution techniques include distribution of operators among the nodes, which makes it possible to scale out with the number of operators by adding more nodes [23]. Furthermore, the queries are composed by operators which can be distributed to one or several nodes, this is referred to as *parallel processing.* A widely used distributed and parallel SPE is Apache Storm [24] which is used as BCStream's SPE. Next section introduces Storm and its basic concepts as well as the motivations why Storm is chosen in this thesis work.

Apache Storm as BCStream's SPE

In order to provide consumers continuous insights of their energy usage and potentially get the involvement of consumers in energy conservation and waste reduction, BCStream needs a powerful SPE which can provide features such as real-time, low latency computation, and scalability with respect to input data volume. There are several distributed stream processing engines available, including S4 [17], Apache Storm [24], StreamInsight [19], IBM InfoSphere Streams [25], etc. For this research, Storm is chosen as BCstream's processing engine because of it is a widely used open-sourced real-time stream processing system. Moreover, the project's supervisors have also had experience working with Apache Storm so it is convenient to gain knowledge and expertise if necessary. Furthermore, Storm also offers features such as fault tolerance and distributed computation, which make it suitable for processing huge amounts of data on different machines. Other useful features, provided by Storm, include:

- Parallel processing - Storm's components can be specified to process in parallel which can enhance the system's performance.
- Storm guarantees that every tuple is processed (at-least-once processing guarantee).
- Storm can be used with any programming language such as Java, Ruby, Python, PHP, etc.

2.3.5 Storm SPE

Storm is a real-time, distributed, fault-tolerant, computation system. Like other SPEs, Storm can process huge amounts of data continuously in real-time. Storm programming model consists of Spouts, Bolts, and Topologies. Storm's concept architecture is described as:

Programming model

Topology is a group of Spouts and Bolts arranged in a Directed Acyclic Graph. A topology starts with one or several Spouts followed by Bolts. A continuous query is defined as a topology (see Section 2.3.3).

Spout is the data source of the topology. A Spout can have one or many data sources and it emits streams of tuples from theses underlying data sources to one or several Bolts.

Bolt consumes and processes tuples emitted from Spouts and emits streams of tuples to downstream Bolts. A bolt represents a stream operator (see Section 2.3.3).

Figure 2.1 shows a sample Storm topology with 1 spouts and 3 bolts.

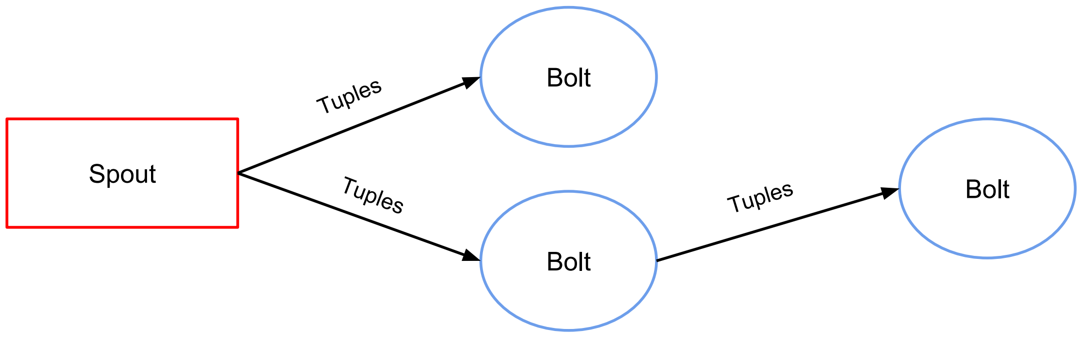


Figure 2.1: Sample Storm topology which consists of one Spout and three Bolts.

Data Flow in a topology

Spout injects streams of data into a topology. A Spout can listen to and pull messages from one or more underlying data sources, when a topology is created it can specify how many tasks to run in parallel for a Spout or a Bolt. Tasks can be viewed as the physical instances of the Spouts and Bolts which can be extracted as logical execution units. The tasks execute the Spout code or Bolt code in parallel in different nodes. A bolt's input is connecting to the output of a Spout or another Bolt.

Storm Concepts

The processing is done in what Storm refers to as workers, and a host machine can have many workers. The workers have executors which are threads spawned by the workers, furthermore the workers then run one or more tasks from either a spout or a bolt (For a more detailed description see Storm documentation [24]).

Fieldsgrouping

Since Storm can operate in parallel and one bolt can be run on one or more tasks at the same time, in order to ensure that one or more data streams always go to the same task, Storm has implemented fieldsgrouping. Fieldsgrouping makes it possible to always send tuples to one specific task based on values from one or more fields.

Fault tolerance & message guarantee

Storm [24] offers fault tolerance such that when a node goes down it is restarted automatically and it is designed to be stateless so a restart of a node would work as if nothing ever happened. Additionally, by offering fault tolerance Storm guarantees all messages to be properly processed within the system. This is done by sending acknowledgements to the spout emitting the tuple the first time when it has reached the final bolt. If the spout does not receive an acknowledgement within the time limit it is considered to be failed, and then it is retransmitted. This feature ensures that all tuples are at least processed once. Since acknowledged messages take up a lot of processing power Storm offers the ability to disable it.

3

System design and architecture

This chapter describes BCStream's architecture and an overview of the modules that compose it. The first part of this chapter presents an overview of BCStream, how its modules are connected, how the data flows through the system and finally how it should be deployed. The following sections continue to describe each module in more detail.

Figure 3.1 shows an overview of BCStream's architecture. BCStream contains three modules: Consumer Configuration System, Stream Processing System, and Publishing System. Additionally, a database is also needed in order to store consumers' information. The first module, Consumer Configuration System, enables consumers to define continuous queries. The queries are then sent to the Stream Processing System. The second module, Stream Processing System, receives energy consumption data from Advanced Metering Infastructures (AMIs see Section 2.2) and processes the data based on the continuous queries defined by the consumers using Apache Storm as SPE (see Section 2.3.4). Lastly, the third module, Publishing System, acquires the query output from the Stream Processing System and sends out feedback to consumers. The Publishing System offers consumers the ability to specify how they want to receive the feedback on social media.

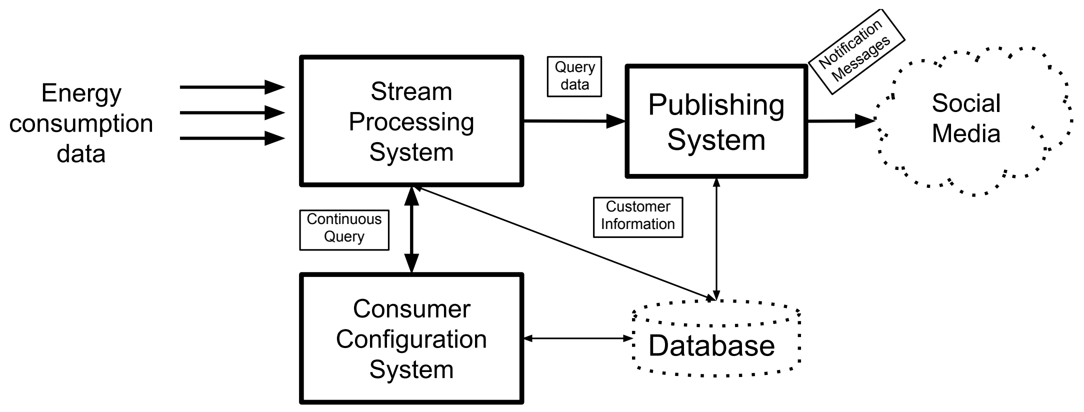


Figure 3.1: An overview of BCStream's architecture.

BCStream's deployment: With fine-grained energy consumption data being available through AMIs and the potential of gigabytes of data packets per second (see 1.1) being generated, the huge amount of data generates even more overhead to the already exhausted networks' bandwidth. BCStream could process data with a higher resolution than the one usually used to bill customers or monitor the network state. Nevertheless, this could not be done in a centralized fashion, because the amount of data could exhaust the infrastructure. Furthermore, it is unnecessary to allow such huge amount of data traveling a long distance through networks to the data center where BCStream is deployed at. To address such problem, the data computation can be distributed to different locations. The idea is to move the computation closer to the data sources such as deploying instances of BCStream at local energy centers for data analysis in different neighborhoods. For example, the total energy consumption of a day and comparison with neighbors can be computed at the local energy center where an instance of BCStream can be deployed at. The fine grained energy readings are not sent further than the local data center and the result is computed and sent back to consumers. Furthermore, the local systems can also send computed data packets to other data centers where other computation can be executed, for instance consumers in different cities want to compare their energy consumption with each other's. The local instance of BCStream located near the households aggregates the energy consumption over a period of time (e.g., one day consumption) and then transmits the result to another data center which handles the computation in that region where the cities are located at. By deploying instances of BCStream at energy centers, we are able to move the data computation closer to the data sources which are consumers' households. Thus we can reduce the amount of data sent through networks. Another advantage of such architecture rather than deploying BCStream in every household is to avoid any modifications of the already deployed metering devices. We refer the readers to Home Energy Management Systems (HEMS) about how analysis could be moved closer to the devices (see [26]).

3.1 Consumer Configuration System

This subsystem provides consumers the ability to compose different queries which are run in the Stream Processing System. Figure 3.2 shows an overview of the Consumer Configuration System. This system provides an Abstract Query Interface where the consumers can compose queries by adding standard operators such as Map, Filter, Aggregate (see Section 2.3.3). A query compiler is utilized to transform an abstract query, composed by consumers, into a Storm topology which is sent to Stream Processing System for deployment.

Furthermore, consumers can also specify how the query output is presented by providing the system their personal information, e.g., a Facebook account. A database stores the obtained information and later on the Publishing System utilizes such data to notify the consumers. The database also stores the configurations which are accessible for the Stream Processing System and the Publishing System.

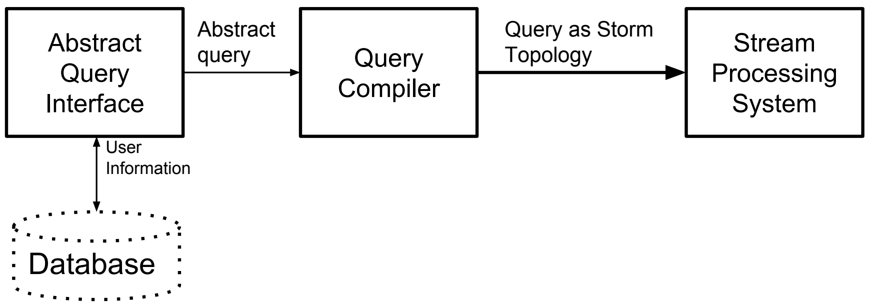


Figure 3.2: Consumer Configuration System overview. The figure indicates the data flow of the Consumer Configuration System.

3.2 Stream Processing System

Stream Processing System consists of two sub-systems which are Data Acquisition System and Stream Processing Engine (SPE) as shown in Figure 3.3. The Data Acquisition System receives energy usage data from AMIs and pushes the data into a queue system such as RabbitMQ [27] or Kafka [28]. The data can be converted, if needed, to a standardized format to make sure that all input messages, as tuples, have the same fields. Then the queue feeds the SPE with data, and in this project the SPE is implemented using Apache Storm [24]. In Storm, data is processed using queries, referred to as topologies (see Section 2.3.5). The SPE receives topologies, defined by consumers, from the Consumer Configuration System. A topology specifies how data is being forwarded and processed using Spouts and Bolts. A Spout connects to the Data Acquisition System, pulls data and emits tuples to the downstream Bolts, which aggregate, process the input data then output results. A topology can consist of several Spouts and Bolts. Depending on what kinds of feedback consumers want, the topology might need to interact with the database for some specific information of consumers, such as home address or number of family members. In addition to using databases, Bolts can also interact with other services like open APIs for energy pricing, weather services, etc. The output from the Bolts is then forwarded to the Publishing System.

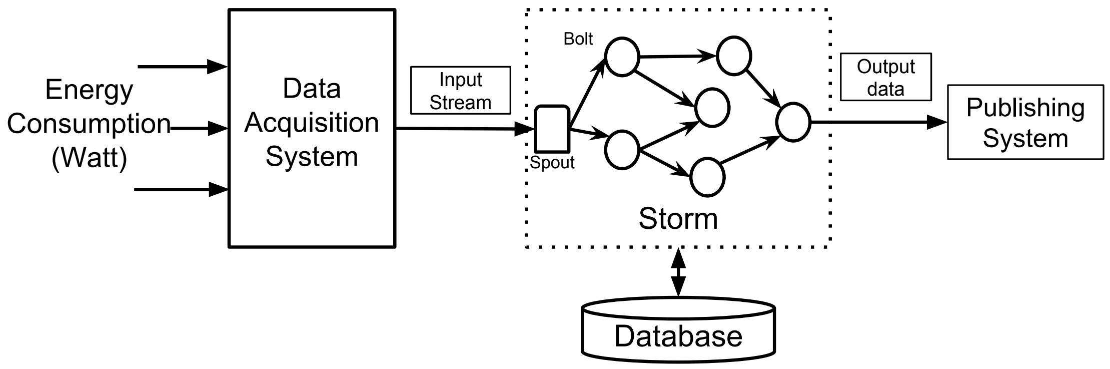


Figure 3.3: Stream Processing System overview. The figure indicates the data flow of the Stream Processing System.

3.3 Publishing System

Information about how feedback is presented on social media is defined by the consumers and stored in a database. By using such information, the Publishing System's task is to make sure that all notifications to consumers are processed properly. Much like the Data Acquisition System in the previous section a queue can be used, where the input data is the output sent from the SPE as can be seen in Figure 3.4. A possible approach is to utilize modular design to have different modules for different notification types, where open-sourced APIs can be used to integrate with, e.g., Facebook and Twitter.

The reason for having Publishing System apart from the Stream Processing System is due to the fact that it is unnecessary to let the Processing System handle all interaction with social media when its main task is to analyze consumption data and should not be interrupted or delayed due to overhead caused by the Publishing System. Furthermore, the Publishing System is not time sensitive, if the interaction with social media is delayed with a couple of seconds or minutes it is not critical.

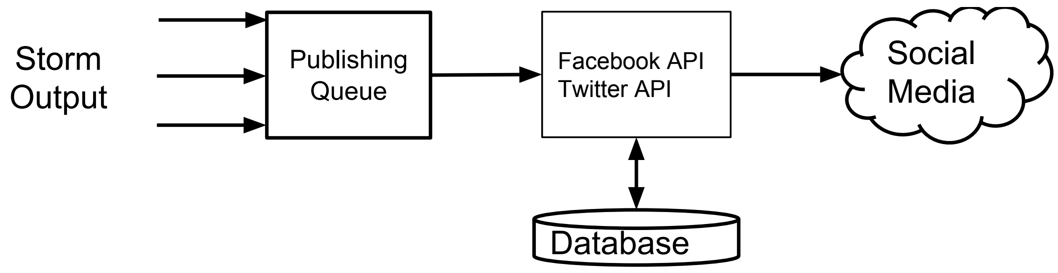


Figure 3.4: Publishing System overview. The figure indicates the data flow of the Publishing System.

As mentioned in Section 2.1.2 the phenomenon of social influence where actions, performed by humans, affect others to act in a similar way is utilized by BCStream. The idea is to present the insights of the consumers' consumption data on social media in a way that other users find it interesting and they start using the same application. A possible approach to intrigue fellow consumers is to create a feeling of a competition, for instance assigning badges and awards, to consumers who are saving a lot of energy, similar to Foursquare [29, 30].

4

Use-cases & Implementation

The main objective of this chapter is to show that standard operators, provided by BCStream, are expressive enough to compose different continuous queries (see Section 2.3.3) for different use-cases. By processing these queries, BCStream provides consumers insights to their energy usage and the opportunity to make proactive actions for saving energy either for environment, economic purpose or just to receive "likes" on social media. In the first section we introduce the schema of the input tuples utilized for data analysis in BCStream. The following sections, we describe three sample use-cases which we have studied, implemented as continuous queries and evaluated (see Section 4.2, 4.3 and 4.4). Using the first use-case, consumers can receive notifications when they use the most energy on daily and weekly basis. The second use-case provides consumers the ability to compare their daily energy consumptions with their neighbours' and determines the top five most consuming neighbours. Lastly, the third use-case, consumers can find out which five electrical appliances consume the most energy in their homes. For each use-case, there is a short overview followed by a more detailed description of how different operators are used to implement the continuous query carrying out the computation. BCStream utilizes Apache Storm [24] to process the continuous queries as its stream processing engine (SPE). However, Storm does not provide operators with the same semantic as BCStream's operators (see Section 2.3.3), and for this reason, we implement the operators in Storm as a prototype. Finally, we present the results of a questionnaire we conducted in order to investigate potential users' opinions on how they would utilize our system (see Section 4.5).

4.1 Input tuple schema

The input tuples represent energy readings, taken every second, from electrical appliances such as PC or TV. Figure 4.2 shows some example energy readings, which contains the timestamp when it is created, the energy consumption (Watt) at that timestamp, the customer which the appliance belongs to and the specific appliance. Readings are taken every second. However some factors like network trouble or faulty appliances might result in some readings not being collected. Furthermore we assume that all tuples entering the system are sorted. The input tuples consumed by all three use-cases share the same schema (Table 4.1).

Field Name	Field Type
Timestamp	*string*
Consumption	*double*
Device	*string*
CustomerID	*string*

Table 4.1: The table represents four fields of the input tuples' schema.

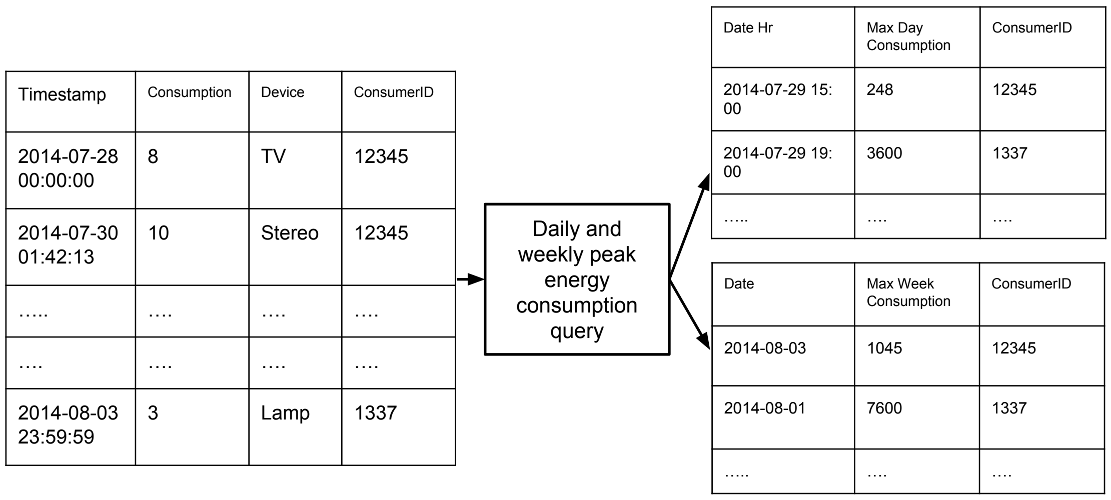
28/12/2011 00:00:01;43;1;TV
28/12/2011 00:00:02;45;3;Lamp
28/12/2011 00:00:04;43;145;TV

Table 4.2: The table shows some sample energy readings taken from electrical appliances.

Field *Timestamp* specifies the timestamp of the tuple while field *Consumption* indicates the consumption in wattage for that second, *Device* determines the specified electronic device and *CustomerID* represents id of the customer whom the tuple refers to.

4.2 Use-case 1: Daily and weekly peak energy consumption

The main objective of this query is to find out consumption peaks and inform the consumers. They receive updates such as which hour of a day and which day in a week the energy is consumed the most and the consumption for that specific time. By receiving such insights on daily and weekly basis, consumers can recognize their power usage patterns and take actions in order to lower their consumption. Moreover, as consumers know their power usage patterns, they can easily discover any abnormal activities which cause more energy consumption than usual and that can not be achieved with the normal monthly utility bills. As shown in Figure 4.1, our system receives input tuples (whose schema is discussed in the previous Section 4.1) containing data such as timestamp, energy consumption, device, customerID. After processing the input tuples using ”Daily and weekly peak energy consumption” query, our system produces two different types of output tuples with schemas are defined as (DateHr, MaxDayConsumption and ConsumerID) and (Date, MaxWeekConsumption and ConsumerID). The output tuples include data such as at what hour of a specific day and which day in a specific week they use the most energy and how much the total consumption is.



Timestamp	Consumption	Device	ConsumerID
2014-07-28 00:00:00	8	TV	12345
2014-07-30 01:42:13	10	Stereo	12345
.....	....	....	....
....	....	....	....
2014-08-03 23:59:59	3	Lamp	1337

Date Hr	Max Day Consumption	ConsumerID
2014-07-29 15: 00	248	12345
2014-07-29 19: 00	3600	1337
.....	....	....

Date	Max Week Consumption	ConsumerID
2014-08-03	1045	12345
2014-08-01	7600	1337
.....	....	....

Figure 4.1: The figure shows some sample input and output of the daily and weekly peak energy consumption query.

4.2.1 Continuous query design

Figure. 4.2 presents the "Daily and weekly peak consumption" use-case as a continuous query using standard operators.

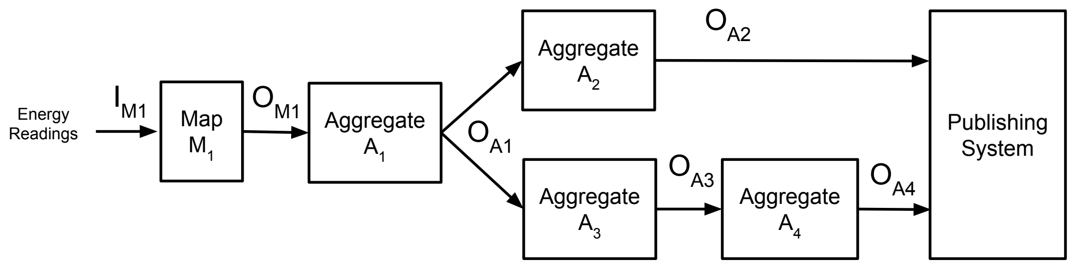


Figure 4.2: Data flow of the Daily and weekly peak energy consumption topology. The figure shows how data streaming operators are used to construct the query.

The query is composed by one stateless operator (Map operator M_1) and four stateful aggregate operators (A_1, A_2, A_3, and A_4).
For each incoming tuple with schema (Timestamp, Consumption, Device, CustomerId) (see Section 4.1), the *Map* operator M_1 removes the field *Device* and generates a output tuple composed by fields (*Timestamp, Consumption and CustomerID*), which are just

copied from the input tuple values. The Device field is removed because it won't be needed by the downstream operators of this query. Operator M_1 is defined as:

M_1*{ Timestamp ← Timestamp, Consumption ← Consumption, CustomerID ← CustomerID },*(I_{M1},O_{M1})

The first *Aggregate* operator A_1 receives tuples produced by *Map* operator M_1. This operator utilizes a tumbling time-based window with window size of one hour to compute the total hourly energy usage by summing the energy consumption from the received tuples within the current window. Furthermore, tuples referring to the same field CustomerID are grouped and processed together to ensure correct computation and a customer's energy consumption can not be altered by others during computation using parameter group-by. The operator A_1 is defined as:

A_1*{time,3600, Date-Hr ← getWindowStart(), totHrConsumption ← sum(Consumption), group-by = CustomerID },*(I_{A1},O_{A1})

The output tuples schema consists of the following fields (*Date-Hr, totHrConsumption and CustomerID*) where field *CustomerID* indicates the consumer identification. Field *totHrConsumption* refers to the total energy usage of a customer using a window of one hour. Field Date-Hr represents the specific hour for which the *totHrConsumption* is measured and it is set to the window start (see Section 2.3.3). These output tuples are emitted to *Aggregate* operators A_2 and A_3. Once the *Aggregate* operator A_2 receives tuples with hourly wattage for a whole day, it determines at which hour the customers use the most power and the results are forwarded to the Publishing System (see Section 3.3) to inform the consumers.

As in the operator A_1, operator A_2 also has a time-based window and parameter group-by on field CustomerID. However, the window size is one day (86400 seconds) instead of one hour. The *Aggregate* operator A_2 is defined as:

A_2*{time,86400, Date-Hr ← getHourWithMaxConsumption(Date-Hr), maxDayConsumption ← max(totHrConsumption), group-by = CustomerID },*(I_{A2},O_{A2})

The output tuples schema is composed by fields (*Date-Hr, MaxDayConsumption and CustomerID*) where field *CustomerID* is the same as the input tuple value produced by *Aggregate* operator A_1. Field *Date-Hr and maxDayConsumption* refer to the hour with the maximum value of energy consumption under a period of one day.

Next, the *Aggregate* operator A_3 receives tuples emitted from operator A_1 every hour. Operator A_3 is almost the same as *Aggregate* operator A_1. The main difference is that A_3 utilizes a window of 24 hours to compute the total power usage instead of one hour. The A_3 operator is defined as:

A_3 *{time,86400, Date ← getWindowStart(), totDayConsumption ← sum(totHrConsumption), group-by = CustomerID },(I_{A3},O_{A3})*

The output tuples schema has the following fields (*Date, totDayConsumption, CustomerID*) where field *CustomerID* is the same as the input tuple value produced by Aggregate operator A_1, field *totDayConsumption* refers to the total energy consumption for one specific day. Field Date specifies which day the consumption is measured and, similar to operator A_1, it is set to the window start.

Finally, the stateful aggregate operator A_4 receives tuples emitted from operator A_3 every day. Operator A_4 is almost the same as aggregate operator A_2. The main difference is that A_4 has a time-based window of one week (604800 seconds). It determines at which day, within a week, the most energy is consumed and also how much energy that is used during that day. Furthermore, it forwards the results to the Publishing System. The A_4 operator is defined as:

A_4*{time,604800, Date ← getDateWithMaxConsumption(TotDayConsumption, Date), maxWeekConsumption ← max(totDayConsumption), group-by = CustomerID },(I_{A4},O_{A4})*

The output tuples schema consists of the following fields (Date, maxWeekConsumption, CustomerID) where field CustomerID is copied from the input tuple values of Aggregate operator A_3, field Date and maxWeekConsumption refer to the day with the maximum value of energy consumption under a period of one week.

4.2.2 Implementation in Storm

For our use-case, we designed one topology of one Spout and five Bolts (see Section 2.3.5) that processes energy readings generated by AMIs to inform consumers their energy usage peaks. In terms of Storm components, the Spout reads the incoming data from the Data Acquisition System (see 3.2), converts them to tuples, and then emits these tuples to downstream Bolts to perform energy consumption peaks analysis.

As shown in Figure 4.3, the InputSpout receives the input data from the Data Acquisition System, converts it to tuples and emits the tuples to the RemoveFieldsBolt (B1), SumBolt_Hour (B2), MaxBolt_Day (B3), SumBolt_Day (B4), MaxBolt_Week (B5) for further processing. Once the processing is done, the consumption peaks are emitted to the Publishing System, where they are published on social media. The detailed implementation for this process is explained next.

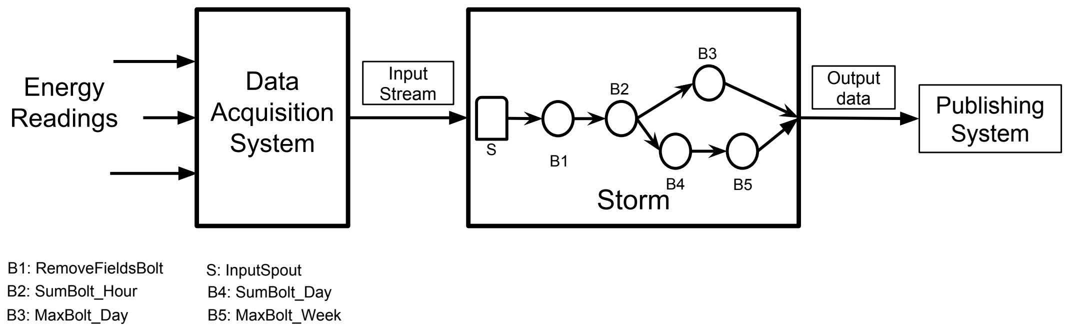


Figure 4.3: Data flow of use-case Daily and weekly peak energy consumption.

InputSpout

As shown in Figure 4.4, InputSpout takes energy readings (Table 4.2), which contains energy data information such as timestamp when the reading is created, the specific device whose consumption is registered, its consumption at that timestamp, and the customer who the device belongs to. Then it generates a tuple for each reading with a schema containing four fields *(Timestamp, Consumption, Device, and CustomerID)* and emits these tuples to downstream Bolts for processing.

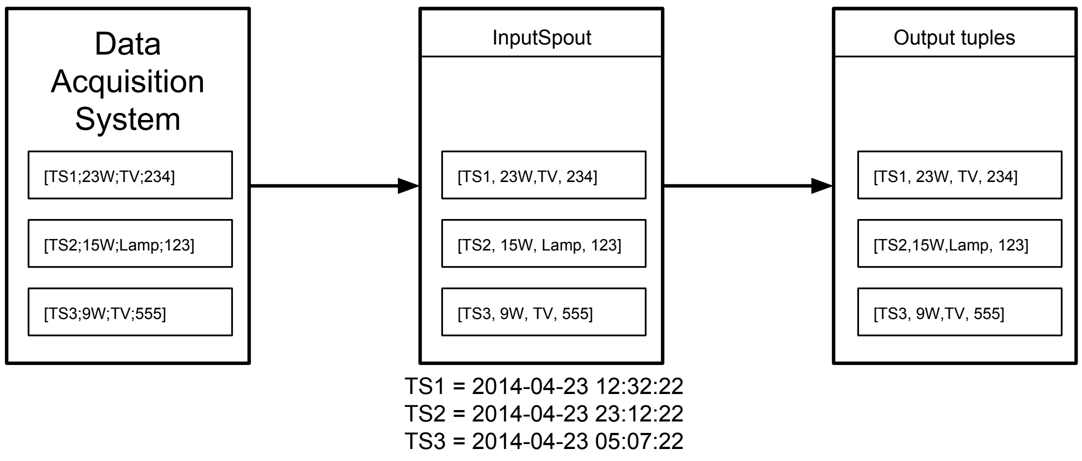


Figure 4.4: Data flow from Data Acquisition System to InputSpout. The figure shows how the inputSpout generates tuples from the input energy readings from the Data Aquisition System.

RemoveFieldsBolt

RemoveFieldsBolt implements the functionality of Map operator M_1 (see Section 4.2.1). It receives one or more parameters which indicate one or more fields to be removed.

In this case, the output tuples of InputSpout are forwarded to RemoveFieldsBolt to remove the unused field Device and produce new tuples with the new schema containing following fields Timestamp, Consumption, CustomerID. The values of the remaining fields stay the same as the input tuples' values. Figure 4.5 shows how RemoveFieldsBolt works.

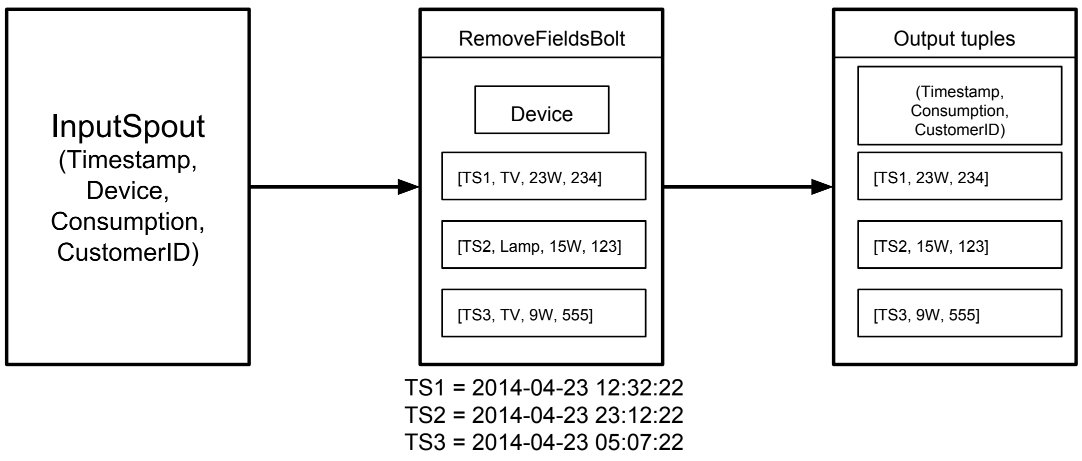

InputSpout
(Timestamp, Device, Consumption, CustomerID)
RemoveFieldsBolt
Device
[TS1, TV, 23W, 234]
[TS2, Lamp, 15W, 123]
[TS3, TV, 9W, 555]
Output tuples
(Timestamp, Consumption, CustomerID)
[TS1, 23W, 234]
[TS2, 15W, 123]
[TS3, 9W, 555]
TS1 = 2014-04-23 12:32:22
TS2 = 2014-04-23 23:12:22
TS3 = 2014-04-23 05:07:22

Figure 4.5: Flow of data from InputSpout to RemoveFieldsBolt. The figure shows that RemoveFieldsBolt removes field Device in input tuples from InputSpout.

The tuples emitted by RemoveFieldsBolt are passed to the next corresponding Bolt, which is a SumBolt in this case.

SumBolt_Hour & SumBolt_Day

SumBolt computes the total energy consumption of a consumer within a predefined tumbling window (see 2.3.3). To implement the Aggregate semantics, a SumBolt requires: what type of window (time-based or tuple-based), window size, what fields of the input tuple schema are used to calculate the sum and a grouping parameter which indicates how tuples should be partitioned among the Bolt's tasks (see Section 2.3.5). Aggregate operators A_1 and A_3 (see Section 4.2.1) are implemented as SumBolt_Hour B2 and SumBolt_Day B4(see Figure 4.4) with time-based windows of one hour and one day respectively. The reason that time-based windows are used in this context is that we are only interested in tuples with timestamps within a certain period of time.

As tuples arrive at SumBolt_Hour B2 from RemoveFieldsBolt (see B1 in Figure 4.4) with timestamps within the current time-based window, SumBolt_Hour B2 does not store every tuple. Instead, it only updates the sum of the power usage stored in field Consumption and then discards the tuples. When a tuple with timestamp outside of its window, the SumBolt_Hour B2 immediately updates its time-based window based on the newly received tuple's timestamp, emits the total energy consumption of the tuples within the previous time-based window and also updates the sum of energy consumption so far with the newly received tuple's value.

As shown in Figure 4.6, SumBolt_Hour B2 receives parameters which indicate that SumBolt_Hour B2 has a time-based window of one hour, it computes the total energy usage in one hour for a consumer using field Consumption of the input tuple schema. Additionally, field CustomerID is used as a grouping rule to ensure that tuples with same CustomerID are always going to the same task. However, tuples with different CustomerID might also end up in the same task so in order to ensure correct computations, a data structure, which stores the customers' energy consumption data in memory, is utilized.

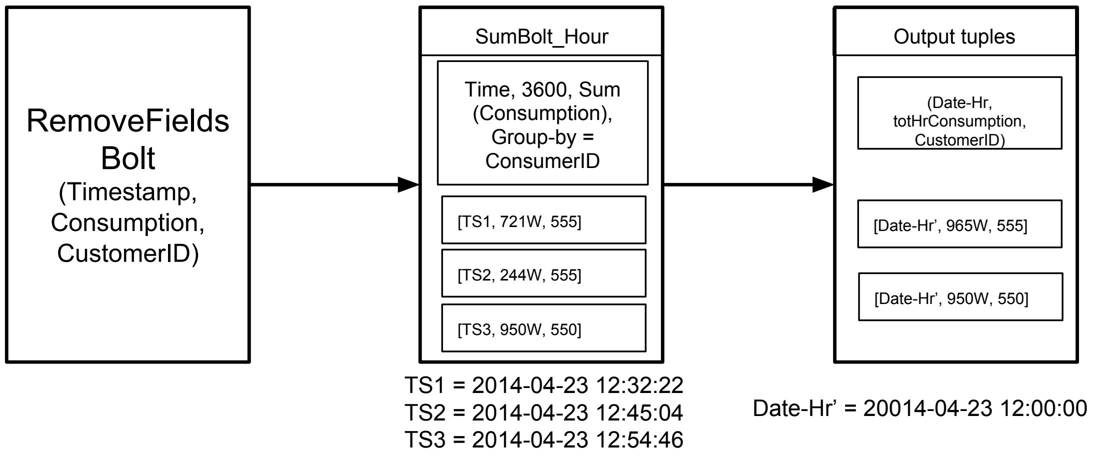

RemoveFields Bolt
(Timestamp, Consumption, CustomerID)
SumBolt_Hour
Time, 3600, Sum (Consumption), Group-by = ConsumerID
[TS1, 721W, 555]
[TS2, 244W, 555]
[TS3, 950W, 550]
Output tuples
(Date-Hr, totHrConsumption, CustomerID)
[Date-Hr', 965W, 555]
[Date-Hr', 950W, 550]
TS1 = 2014-04-23 12:32:22
TS2 = 2014-04-23 12:45:04
TS3 = 2014-04-23 12:54:46
Date-Hr' = 20014-04-23 12:00:00

Figure 4.6: Data flow from RemoveFieldsBolt to SumBolt_Hour. The SumBolt summarizes the consumption over an hour produces output tuples.

SumBolt_Day B4 (4.7) receives a tuple with schema *(Date-Hr, totHrConsumption, CustomerID)* (see Figure 4.6) from SumBolt_Hour every hour. It is implemented almost exactly the same as SumBolt_Hour B2. B4 has a time-based window of one day (86 400 seconds) instead of one hour. It also uses field *totHrConsumption* in order to compute the total energy usage for a day and field *ConsumerID* to partition stream of tuples into its tasks

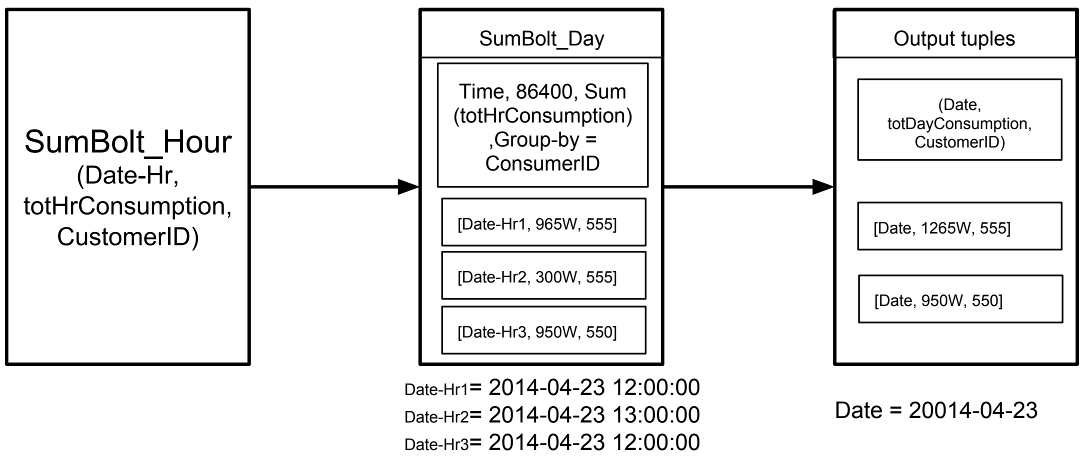

SumBolt_Hour
(Date-Hr, totHrConsumption, CustomerID)
SumBolt_Day
Time, 86400, Sum (totHrConsumption) ,Group-by = ConsumerID
[Date-Hr1, 965W, 555]
[Date-Hr2, 300W, 555]
[Date-Hr3, 950W, 550]
Output tuples
(Date, totDayConsumption, CustomerID)
[Date, 1265W, 555]
[Date, 950W, 550]
Date-Hr1= 2014-04-23 12:00:00
Date-Hr2= 2014-04-23 13:00:00
Date-Hr3= 2014-04-23 12:00:00
Date = 20014-04-23

Figure 4.7: Data flow from SumBolt_Hour to SumBolt_Day which summerizes the total energy consumption of a specific customer under one day and generates output tubles based on the input tubles from the SumBolt_Hour.

MaxBolt_Day & MaxBolt_Week

MaxBolts determine the maximum energy usage of a consumer within a certain window. Similar to SumBolt, a MaxBolt requires several parameters: window, a window size, what fields of the input tuple schema to calculate the maximum energy usage within the window and a grouping parameter which indicates how tuples should be sent to the Bolt's tasks (see Section 2.3.5). *Aggregate* operators A_2 and A_4 (see Section 4.2.1) are implemented as MaxBolt_Day B3 and MaxBolt_Week B5(see Figure 4.4) with time-based windows of one day and one week respectively. The reason that time-based windows are used in these Bolts is that we are only interested in finding out which hour of a day and which day of a week the consumers use the most energy.

Figure 4.8 shows the data flow from SumBolt_Day through MaxBolt_Day B3. A tuple is sent from SumBolt_Day to MaxBolt_Day every hour with a schema *(Date-Hr, totHrConsumption, CustomerID).* MaxBolt_Day has a time-based window of one day (86 400 seconds) and uses field *Date-Hr* and *totHrConsumption* to determine the hour in which the most energy is consumed. Moreover, field *ConsumerID* is once again used as stream grouping rule to ensure that tuples with same *CustomerID* are always going to the same task. Instead of storing every input tuple, MaxBolt_Day utilizes a simple data structure (for instance Map) to store consumers' total hourly consumption data in memory. Since tuples with different *CustomerID* might end up in the same task so in order to ensure the computation of energy consumption for a customer would not be altered by other consumers, the data structure is needed.

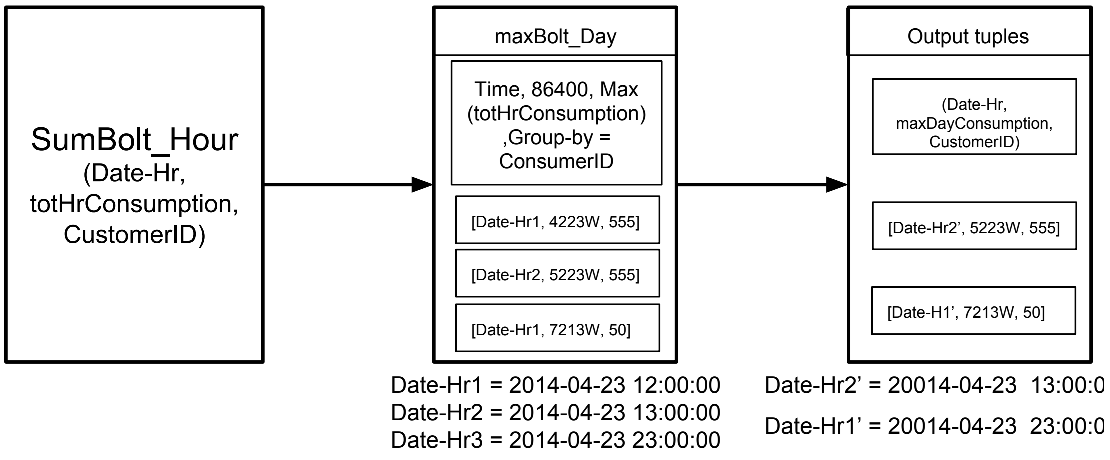

SumBolt_Hour
(Date-Hr, totHrConsumption, CustomerID)
maxBolt_Day
Time, 86400, Max (totHrConsumption) ,Group-by = ConsumerID
[Date-Hr1, 4223W, 555]
[Date-Hr2, 5223W, 555]
[Date-Hr1, 7213W, 50]
Output tuples
(Date-Hr, maxDayConsumption, CustomerID)
[Date-Hr2', 5223W, 555]
[Date-H1', 7213W, 50]
Date-Hr1 = 2014-04-23 12:00:00
Date-Hr2 = 2014-04-23 13:00:00
Date-Hr3 = 2014-04-23 23:00:00
Date-Hr2' = 20014-04-23 13:00:0
Date-Hr1' = 20014-04-23 23:00:0

Figure 4.8: Flow of data from SumBolt_Hour to MaxBolt_Day. The MaxBolt_Day computes under which hour of a day, a customer consumes the most energy and generates output tubles based on the input tubles from the SumBolt_Hour.

MaxBolt_Week receives a tuple from SumBolt_Day with a tuple schema *(Date, totDayConsumption, CustomerID)* everyday (see Figure 4.9). The implementation is almost exactly the same as SumBolt_Hour. MaxBolt_Week has a time-based window of one week (604800 seconds) instead of one day. It also uses field *totDayConsumption* and *Date* in order to determine in which day the most energy is used and field *ConsumerID* to partition stream of tuples into its tasks.

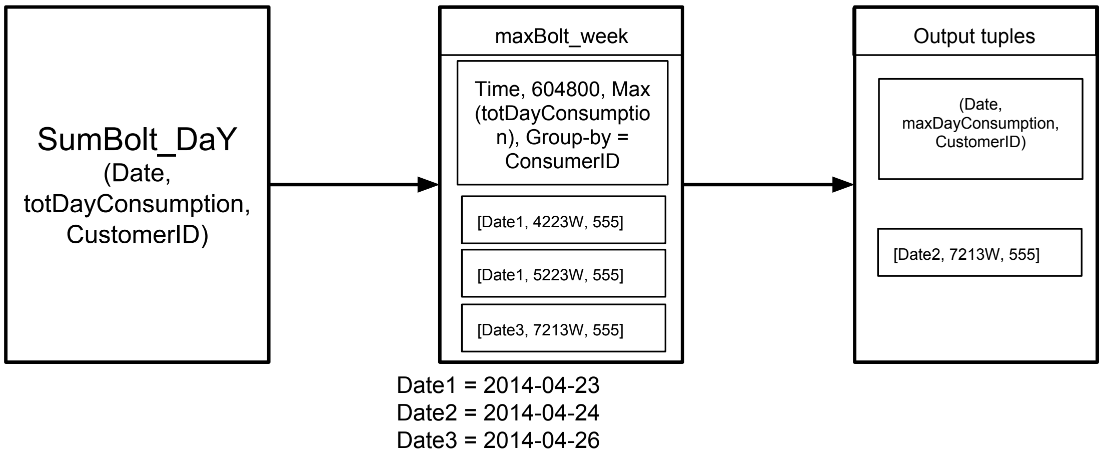
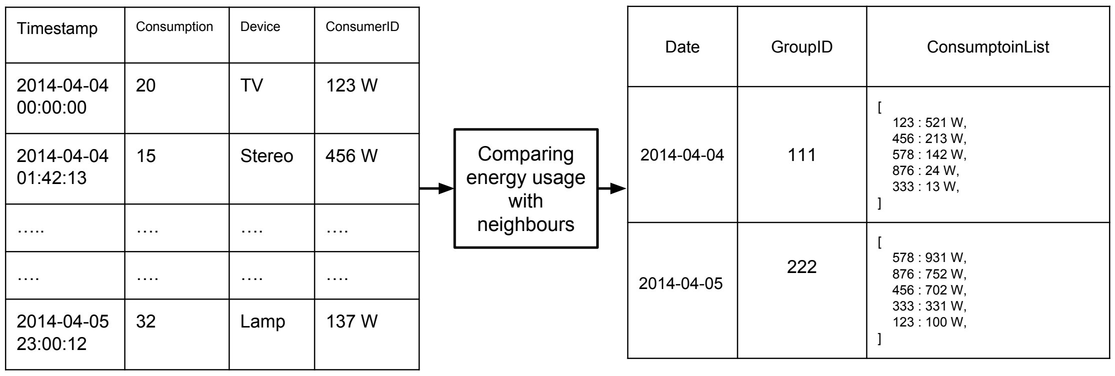


Figure 4.9: Data flow from SumBolt_Day to MaxBolt_Week. MaxBolt_Week outputs the day with the highest consumption for that week.

4.3 Use-case 2: Comparing energy usage with neighbours

This second use-case enables consumers to compare their consumption with each other, whether it is neighbors, friends or both. The purpose is to give a feeling of a competition and by that intrigues and influences people to consume less energy. This use-case determines the top five consumers per day, however it could be extended to work with any given time period, e.g., week, month, etc. The output of this use-case is be a list of consumers and their daily consumption which is sent to the Publishing System (see Section 3.3 for distribution of the data. Figure 4.10 gives an example of how the input and the expected output tuples look like.



Timestamp	Consumption	Device	ConsumerID
2014-04-04 00:00:00	20	TV	123 W
2014-04-04 01:42:13	15	Stereo	456 W
.....	....	....	....
....	....	....	....
2014-04-05 23:00:12	32	Lamp	137 W

Date	GroupID	ConsumptoinList
2014-04-04	111	[123 : 521 W, 456 : 213 W, 578 : 142 W, 876 : 24 W, 333 : 13 W,]
2014-04-05	222	[578 : 931 W, 876 : 752 W, 456 : 702 W, 333 : 331 W, 123 : 100 W,]

Figure 4.10: The figure shows some sample input and output of the Comparing energy usage with neighbours use-case.

4.3.1 Continuous query design

Figure 4.11 shows use-case Comparing energy usage with neighbours as a continues query composed by standard data streaming operators (see 2.3.3). The query consists of two stateless Map operators and two stateful aggregate operators (A_3 and A_5). Operators M_1 and A_3 works just like the ones used in the previous use-case (see Section 4.2.1).

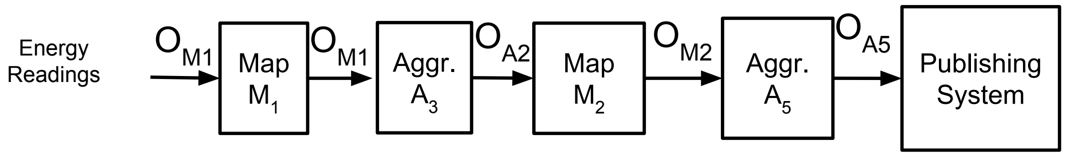


Figure 4.11: Comparing energy usage with neighbours topology. The figure shows how data streaming operators are used to construct the query.

The *Map* operator M_1 removes the field *Device* since this use-case only calculates the daily and weekly peak consumptions of a household. The main reason is to reduce the computational cost of the query by removing unnecessary information that are not used to compute the query result. The operator M_1 is defined as:

M1{ Timestamp ← Timestamp, Consumption ← Consumption, CustomerID ← CustomerID },(I,O_{M1})

Aggregate operator A_3 receives tuples from Map operator M_1 and are grouped by the field CustomerID. A_3 uses a time-based window and sums up the consumption over the window size of 86400 seconds to compute the daily consumption and outputs the result. Consumption is summed up and a new tuple is output with a schema consisting of the following fields: *Date, totDayConsumption and CustomerID* and Date is set to window start. The operator A_3 is defined as:

A_3{*time,86400, Date ← getWindowStart(), totDayConsumption ← sum(totHrConsumption), group-by = CustomerID* },(I_{A3},O_{A3})

The operator M_2 which receive tuples from A_3 with the daily total consumption. M_2 adds the field GroupID to the tuples in order to find out which consumers to compare with. The function getGroupID gives the groupID based on the CustomerID and is used in A_5 with the group by parameter. The output tuple schema consists of the following fields: *Date, totDayConsumption , CustomerID, GroupID* and the operator is defined as:

M_2{*Date ← Date, totDayConsumption ← totDayConsumption, CustomerID ← CustomerID, GroupID ← getGroupID(customerID)* },(I_{M2},O_{M2})

Finally the operator A_5 receives tuples from M_2 and determine the top five consumers over the time window of one day. The parameter group by is used on GroupID to only aggregate over the consumers that are in the same group. The function topConsumers then returns a list containing the consumption and CustomerID of the top consumers for that group. The output tuple schema consists of the following fields: *Date, CompareList, GroupID* and the operator is defined as:

A_5{*time,86400, Date ← getWindowStart(), CompareList ← topConsumers(totDayConsumption, CustomerID) , group-by = GroupID* },(I_{A5},O_{A5})

4.3.2 Implementation in Storm

For the use-case Comparing energy usage with neighbours, we designed one topology of one Spout and four Bolts (see Section 2.3.5). The Spout reads the incoming data from the Data Acquisition System (see 3.2), after converting it to tuples with a schema consisting of several fields. Furthermore, the spout emits the tuples to the downstream Bolts to perform energy consumption comparison with different consumers.

Figure 4.12, much like the previous use-case (see Section 4.2.2) the InputSpout accepts the input data from the Data Acquisition System, convert it to tuples and emits the tuples to the RemoveFieldsBolt (B1) and MaxBolt_Day (B2) for further processing. Furthermore, there are 2 other bolts introduced in this section: AddFieldsBolt and TopConsumersBolt_Day which are described in more detail.

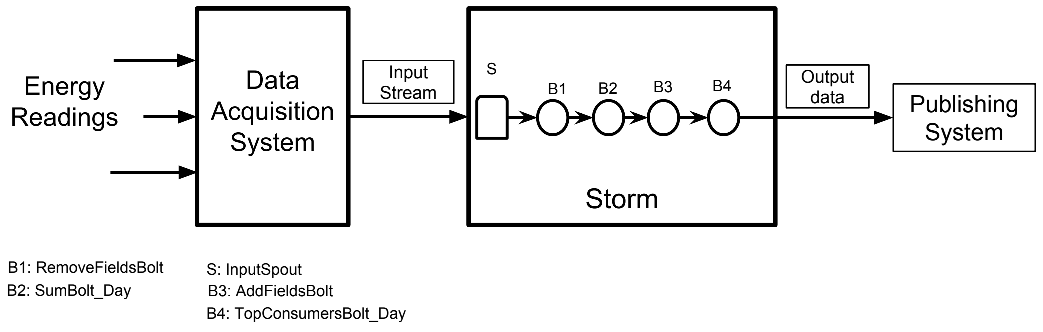


Figure 4.12: Data flow for use-case Comparing energy usage with neighbours.

AddFieldsBolt

This Bolt is the implementation of the operator M_2 described earlier in this section and the purpose is to add the field GroupID to the tuples. The field is then used as input for the group by parameter in the next bolt, TopConsumersBolt_Day and it ensures that consumers belonging to the same group are compared together. All tuples entering this bolt invoke a call to the function getGroupID(CustomerID) which returns all the groups the consumer of that tuple belongs to. For each group, a new tuple with the a new schema consisting of the fields: Date, totDayConsumption , CustomerID and GroupID is emitted, some example tuples are shown in Figure 4.13. The function getGroupID has access to the database where the consumer information, regarding what groups to compare with, is stored.

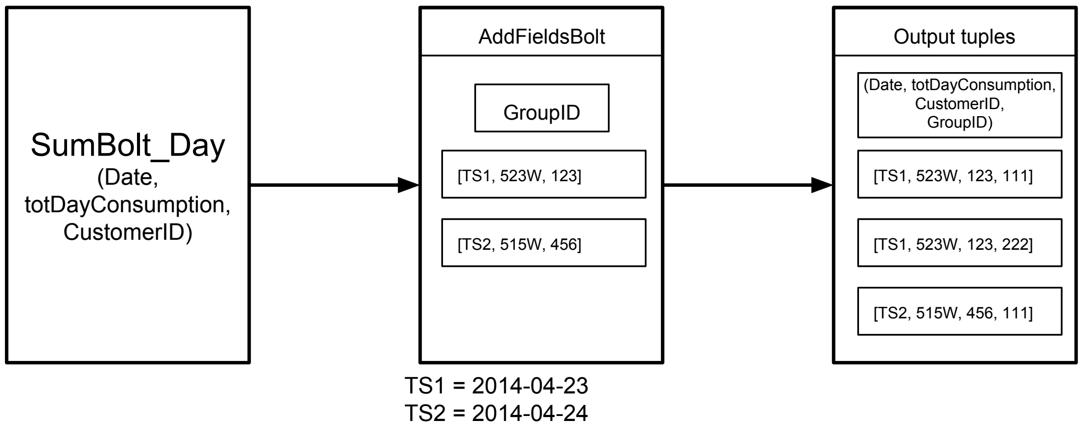


Figure 4.13: Data flow from SumBolt_Day to AddFieldsBolt. The Figure shows how AddFieldsBolt generate output tuples containg an extra field.

TopConsumersBolt_Day

This bolt correlates to the operator A_5 and determines the top five most energy consuming households within a group (neighbourhood, Facebook friends, etc.,). This bolt uses a time-based window with a window size of one day. TopConsumerBolt_Day receives tuples emitted by AddFieldsBolt and stores the consumption and CustomerID in a data structure and then discards the tuple. To make sure that all users within the same comparing group ends up in the same task, GroupID is used as a fieldsGrouping (see Section 2.3.5) parameter for this bolt. When a tuple with a timestamp greater than windows start + window size is received the bolt calls a function topConsumers. TopConsumers then computes which the top most consuming households are and returns a list (CompareList) containing the CustomerIDs and their daily consumption. TopConsumers function takes a parameter MaxConsumers which indicates the maximum number of people that should be in the list which is set to five in this use-case. Finally when the list is done, this Bolt emits a new tuple consisting of the fields: Date, CompareList and GroupID. Figure 4.14 shows some example input and output tuples:

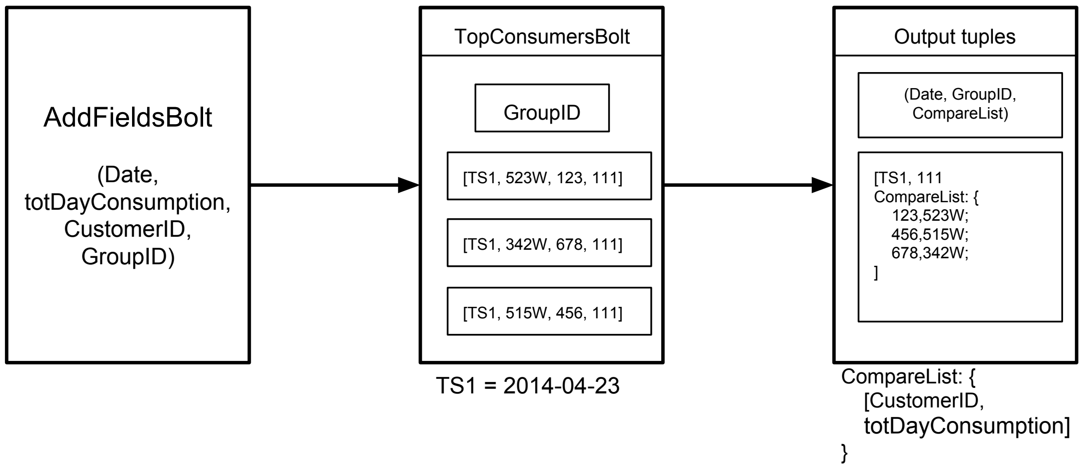
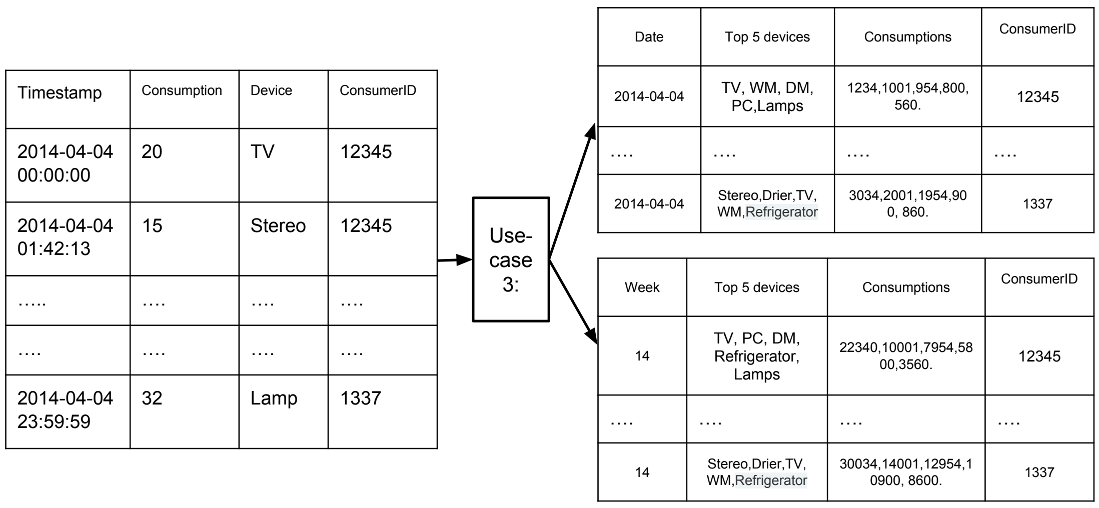

AddFieldsBolt
(Date, totDayConsumption, CustomerID, GroupID)
TopConsumersBolt
GroupID
[TS1, 523W, 123, 111]
[TS1, 342W, 678, 111]
[TS1, 515W, 456, 111]
TS1 = 2014-04-23
Output tuples
(Date, GroupID, CompareList)
[TS1, 111 CompareList: { 123,523W; 456,515W; 678,342W;]
CompareList: { [CustomerID, totDayConsumption] }

Figure 4.14: Flow of data from AddFieldsBolt to TopConsumersBolt. The Figure shows how TopConsumerBolt find the top 5 consumers and outputs it.

4.4 Use-case 3: Top five energy consumption appliances

The main objective of this use-case is to find out and inform the consumers which five power consumption appliances consume the most power on daily and weekly basis. By receiving such insights, consumers can easily understand their households' energy usage, especially how much power each appliance actually consumes. They might take appropriate actions such as replacing an old refrigerator or a washing machine which consumes too much power. Consequently, it improves the energy saving either for the economic or environment purpose. Figure 4.15 shows how our system receives tuples with schema (Timestamp, Consumption, Device, Customer) (see Section 4.1). After processing the input data, our system produces outputs including top five energy consuming appliances and the consumption of each appliance.



Timestamp	Consumption	Device	ConsumerID
2014-04-04 00:00:00	20	TV	12345
2014-04-04 01:42:13	15	Stereo	12345
…..	….	….	….
….	….	….	….
2014-04-04 23:59:59	32	Lamp	1337

Date	Top 5 devices	Consumptions	ConsumerID
2014-04-04	TV, WM, DM, PC,Lamps	1234,1001,954,800, 560.	12345
….	….	….	….
2014-04-04	Stereo,Drier,TV, WM,Refrigerator	3034,2001,1954,90 0, 860.	1337

Week	Top 5 devices	Consumptions	ConsumerID
14	TV, PC, DM, Refrigerator, Lamps	22340,10001,7954,58 00,3560.	12345
….	….	….	….
14	Stereo,Drier,TV, WM,Refrigerator	30034,14001,12954,1 0900, 8600.	1337

Figure 4.15: The figure shows some sample input and output of the Top five energy consumption appliances query.

4.4.1 Continuous query design

Figure 4.16 introduces use-case Top five energy consumption appliances as a continuous query composed by standard operators.

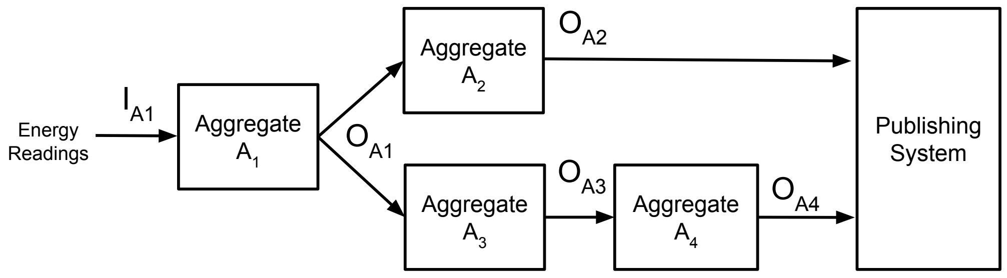


Figure 4.16: Data flow of the Top five energy consumption appliances topology. The figure shows how data streaming operators are used to construct the query.

The query is composed by four stateful aggregate operators (A_1, A_2, A_3 and A_4). The incoming tuples are consumed by the aggregate operator A_1. This operator utilizes a tumbling time-based window of one day (expressed in seconds in the operator definition). It uses a sum function to compute the total consumption of an appliance using data in field Consumption. The operator is defined as:

$A_1\{time,86400,\ Date \leftarrow getWindowStart(),\ Device \leftarrow Device,\ totDayConsumption \leftarrow sum(Consumption),\ group\text{-}by = CustomerID\ \},(I_{A1},O_{A1})$

The output tuples schema is composed by fields (*Date, Device, totDayConsumption and Customer*) where field CustomerID is copied from the input tuple values and field totDayConsumption indicates the total energy usage of a customer under a specific day. In addition, field Date is set to the window start and refers to the day in which the totDayConsumption is computed. These output tuples are consumed by aggregate operators A_2 and A_3. Once the Aggregate operator A_2 receives tuples with the daily wattage of every energy consumption appliance for a whole day (86400 seconds), it determines a list of five appliances which consume the most energy in the previous day and the result is sent to Publishing System (see Section 3.3) to inform the consumers. The aggregate operator A_2 is defined as:

A_2*{time,86400, Date ← getWindowStart(), ConsumptionList ← top5Devices(totDayConsumption, device), DeviceList ← top5Devices(totDayConsumption,device), group-by = CustomerID },(*I_{A2}*,*O_{A2}*)*

The output tuples schema is composed by fields (*Date, ConsumptionList, DeviceList, CustomerID*) where field CustomerID is the same as the input tuple value. Field ConsumptionList and DeviceList indicate the top five appliances and their energy consumption values. Finally, field Date determines under which date the result is produced.

Operator A_3 receives tuples emitted by A_1 every day. This operator uses a time-based window of a week (expressed in seconds in the definition) and calculates the top five consuming devices. The operator A_3 is defined as:

A_3*{time,604800, week ← getWeek(), Device ← Device, totWeekConsumptions ← sum(totDayConsumption), group-by = CustomerID }, (*I_{A3}*,*O_{A3}*)*

The output tuple schema consist of 4 fields: *(week, Device, totWeekConsumption and-CustomerID)*, fields Device and CustomerID are just copied from the input tuple. Field totWeekConsumption refers to the total energy consumption of a specific device under a specific week. The Week field determines the week number of the current window

Finally, operator A_4 is very similar to A_2 with the only difference that the operator A_4 determines the top five consuming devices under a week (604800 seconds) instead of one day as in operator A_3. In addition, the result is forwarded to the Publishing System to inform the consumers. The operator A_4 is defined as:

A_4*{time,604800, week ← getWeek(), ConsumptionList ← top5Devices(totWeekConsumption, device), DeviceList ← top5Devices(totWeekConsumption, device), group-by = CustomerID },(*I_{A7}*,*O_{A7}*)*

The output tuple schema consist of 4 fields: *(Week, DeviceList, ConsumptionList, Cus-*

tomerID), whereas field CustomerID is just copied from the input tuple. Fields DeviceList, ConsumptionList, and Week refers to the total energy consumptions of top five devices under a specific week.

4.4.2 Implementation in Storm

For this use-case we define a topology consisting of one Spout and four Bolts. Figure 4.17 shows how the InputSpout receives energy readings from the Data Acquisition System, converts them to tuples (whose schema is discussed in Section 4.1) and finally emits those newly created tuples to downstream Bolts for further processing. Once the analysis is done, a list of top five devices and their daily and weekly energy consumption are sent to the Publishing System, where they are sent to consumers either via e-mail, SMS or on social media such as Facebook or Twitter. The detailed implementation is explained next.

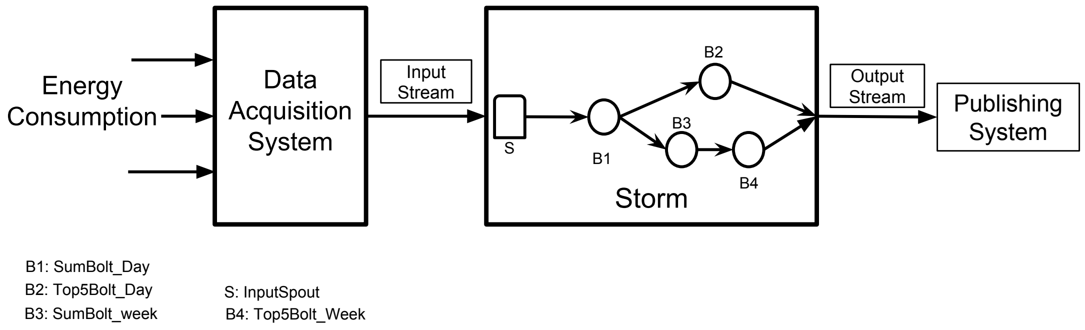


Figure 4.17: Data flow of use-case Top five energy consumption appliances.

InputSpout Implementation

InputSpout's implementation is exactly the same as in the previous use-cases (see Section 4.2.2 for more details).

SumBolt_Hour & SumBolt_Day

Aggregate operators A1 and A3 are implemented as SumBolt_Hour B1 and SumBolt_Day B3 with time-based windows of one day and one week respectively. The implementation is very much the same as in the previous use-cases. SumBolt_Hour B1 (see Figure 4.18) consumes tuples from the InputSpout. As the tuples arrive with timestamps within the current time-based window, SumBolt_Hour B1 updates the total energy usage for every device using data from fields Consumption and Device and then discards the tuple. When a tuple with timestamp outside of its window, the SumBolt_Hour B1 immediately updates its time-based window based on the newly received tuple's timestamp. Then it emits a tuple to downstream Bolts for further analysis. The output tuples schema is defined by following fields (Date, Device, totDayConsumption, CustomerID) whereas field totDayConsumption prefers the total energy usage under a specific period (Date) using a sum function on data contained by field Consumption. Fields Device and CustomerID are the same as the input tuple value. In addition, field CustomerID is used as a grouping rule to ensure that tuples with same CustomerID are always going to the same task. However, tuples with different CustomerID might also end up in the same task so in order to ensure the computation of energy consumption for a customer would not be modified by others, a data structure, which stores the customers' devices and their consumption data, is needed.

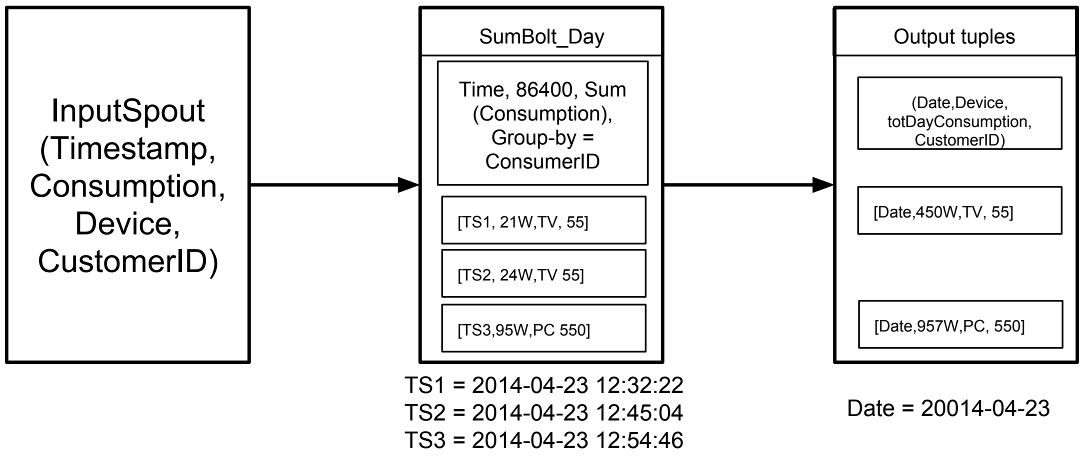

InputSpout (Timestamp, Consumption, Device, CustomerID)
SumBolt_Day
Time, 86400, Sum (Consumption), Group-by = ConsumerID
[TS1, 21W,TV, 55]
[TS2, 24W,TV 55]
[TS3,95W,PC 550]
Output tuples
(Date,Device, totDayConsumption, CustomerID)
[Date,450W,TV, 55]
[Date,957W,PC, 550]
TS1 = 2014-04-23 12:32:22
TS2 = 2014-04-23 12:45:04
TS3 = 2014-04-23 12:54:46
Date = 20014-04-23

Figure 4.18: Flow of data from InputSpout to SumBolt_Day which summerizes different electrical devices' total energy consumption of a specific consumer during one day and generates output tubles based on the input tubles from InputSpout.

SumBolt_Day B3 (Figure 4.19) receives a tuple with schema (Date, totDayConsumption and CustomerID) from SumBolt_Day every day. It is implemented almost exactly the same as SumBolt_Day B1. B3 has a time-based window of one week (604800 seconds) instead of one day. It also uses field totDayConsumption in order to compute the total energy usage for a week and field ConsumerID to partition tuples into its tasks to ensure the tuples with the same CustomerID end up in the same task.

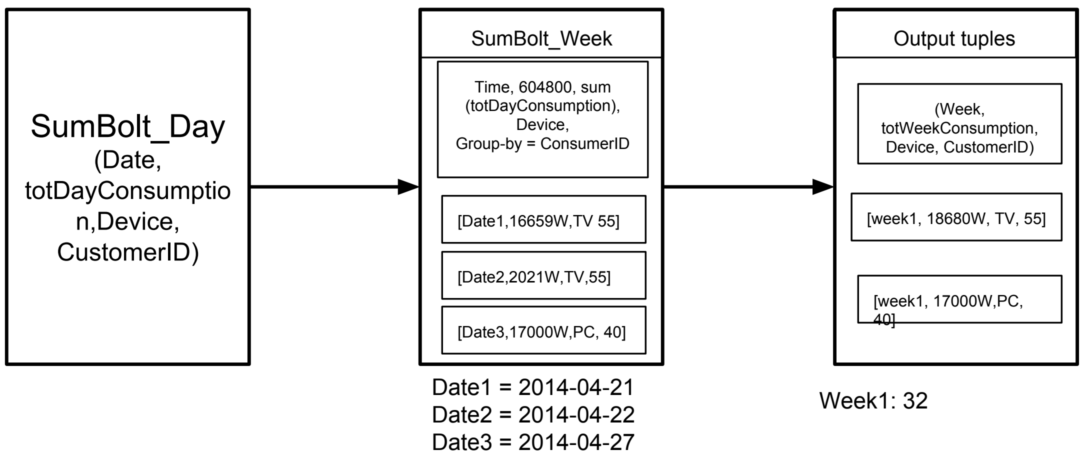


Figure 4.19: Flow of data from SumBolt_Day to SumBolt-Week which summerizes different electrical devices' total energy consumption of a specific consumer during a week and generates output tubles based on the input tubles from the SumBolt_Day.

Top5Bolt_Day & Top5Bolt_Week

Top5Bolts are used to determine the top five energy consuming devices of a consumer within a certain window. Similar to SumBolt, a Top5Bolt requires several parameters: window (either time-based or tuple-based), a window size, a function that determines top five devices that consume the most energy under the predefined window and a grouping parameter which indicates how tuples should be sent to the Bolt's tasks. As tuples with timestamps within the current window arrive, the consuming devices along with their consumption data is stored. When a tuple with timestamp outside of the current window arrives, the function to determine which five devices consume the most power is triggered alongside with their consumption.

Aggregate operators A2 and A4 are implemented as Top5Bolt_Day B3 and Top5Bolt_Week B5 with time-based windows of one day and one week respectively. Figure 4.20 shows the data flow from SumBolt_Day B1 through Top5Bolt_Day B3. A tuple is sent from SumBolt_Day to Top5Bolt_Day every hour with a schema (Date, totDayConsumption, Device and CustomerID). Top5Bolt_Day has a time-based window of one day (86 400 seconds) and uses field totDayConsumption to determine the five appliances which consume the most energy under that specific day. Moreover, field ConsumerID is once again used as stream grouping rule to ensure that tuples with same CustomerID are always going to the same task. Instead of storing every input tuple, Top5Bolt_Day utilizes a simple data structure (e.g., Map) to store consumers' devices and their hourly consumption data in memory. Since tuples with different CustomerID might end up in the same task so in order to ensure the computation of energy consumption for a customer would not be altered by other consumers, the data structure is needed. The results including fields (Date, DeviceList, ConsumptionList and CusomterID) are then forwarded to the Publishing System to inform the consumers.

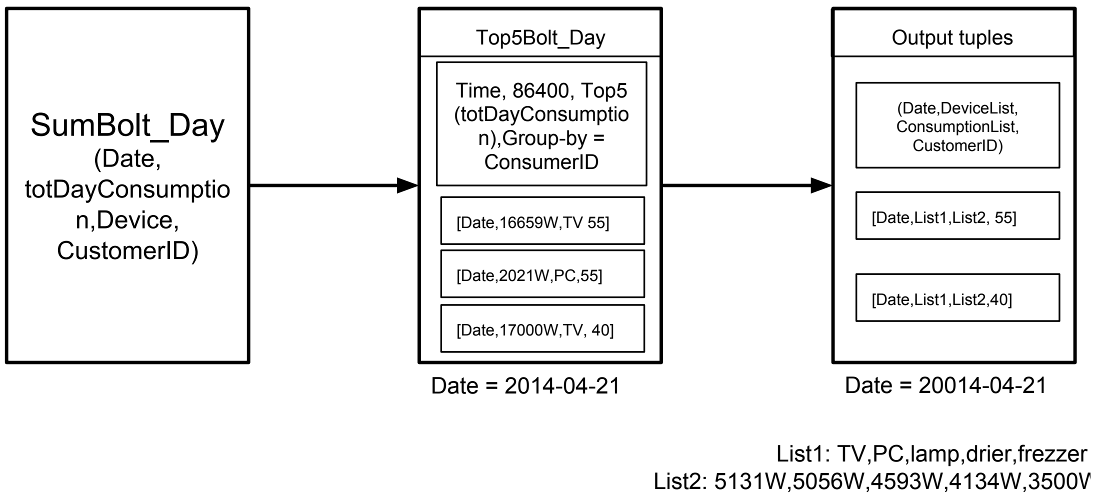

SumBolt_Day
(Date, totDayConsumption,Device, CustomerID)
Top5Bolt_Day
Time, 86400, Top5 (totDayConsumption),Group-by = ConsumerID
[Date,16659W,TV 55]
[Date,2021W,PC,55]
[Date,17000W,TV, 40]
Date = 2014-04-21
Output tuples
(Date,DeviceList, ConsumptionList, CustomerID)
[Date,List1,List2, 55]
[Date,List1,List2,40]
Date = 20014-04-21
List1: TV,PC,lamp,drier,frezzer
List2: 5131W,5056W,4593W,4134W,3500V

Figure 4.20: Flow of data from SumBolt_Day to Top5Bolt_Day which indicates top five electrical devices using the most energy and their energy consumption under one day and generates output tubles based on the input tubles from the SumBolt_Day.

Top5Bolt_Week receives a tuple from SumBolt_Day with a tuple schema (Date, totDay-Consumption, Device and CustomerID) everyday (see Figure 4.21). The implementation is almost exactly the same as Top5Bolt_Day. Top5Bolt_Week has a time-based window of one week (604800 seconds) instead of one day. It also uses field totDayConsumption in order to determine in which five devices consume the most energy during that week and field ConsumerID is utilized to partition incoming tuples into its tasks. The output tuple schema includes following fields (Week, DeviceList, ConsumptionList and CustomerID) is then sent to the Publishing System to inform consumers.

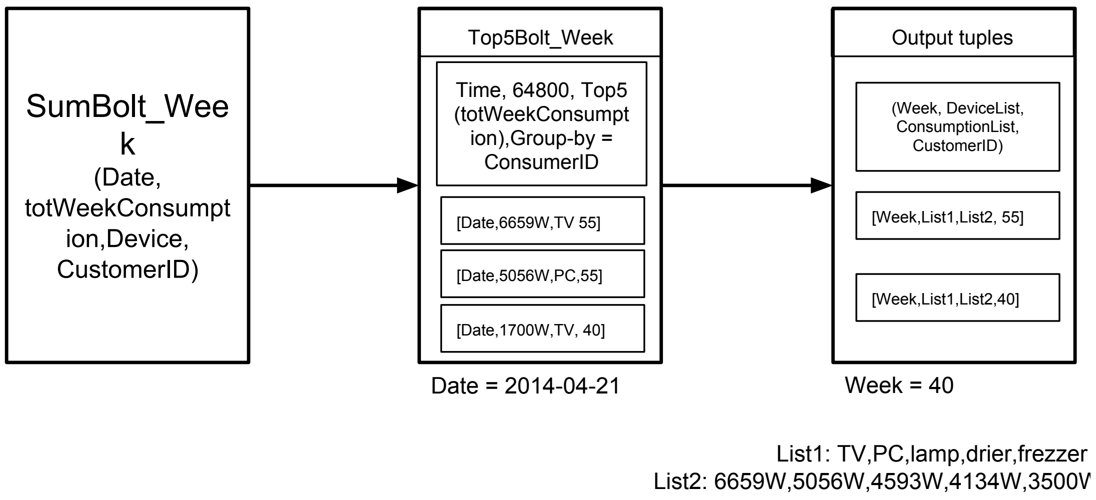

SumBolt_Week (Date, totWeekConsumption,Device, CustomerID)
Top5Bolt_Week
Time, 64800, Top5 (totWeekConsumption),Group-by = ConsumerID
[Date,6659W,TV 55]
[Date,5056W,PC,55]
[Date,1700W,TV, 40]
Date = 2014-04-21
Output tuples
(Week, DeviceList, ConsumptionList, CustomerID)
[Week,List1,List2, 55]
[Week,List1,List2,40]
Week = 40
List1: TV,PC,lamp,drier,frezzer
List2: 6659W,5056W,4593W,4134W,3500V

Figure 4.21: Flow of data from SumBolt_Week to Top5Bolt_Week which indicates top five electrical devices using the most energy and their energy consumption under one week and generates output tubles based on the input tubles from the SumBolt_Week

4.5 Questionnaire

In order to monitor the consumers' expectations of our system, we performed a questionnaire. The results are utilized as complements to develop and implement different real-world use-cases using our system. Special emphasis is given to university students and professors as the target group. The questionnaire starts with a short description of this master thesis as a system which processes real-time energy consumption data to provide consumers feedback of their energy usage using social media. Furthermore, a simple example use-case is also stated in order to make it easier to understand the usage of our system. In the following section we present the questionnaire as well as the response proveded by the participants.

the questions asked in the questionnaire as well as the results are described.

4.5.1 Questions

From the very beginning of the questionnaire planning process, selection of the questions stated in the questionnaire is tied very precisely to the use-cases that would be made of the answers. Following questions are asked:

- If your household contains technology that logs the energy consumption of the electrical appliances, what feedback would you like to receive and how frequently?
- If the system could be integrated with social media, what kind of information would you like to share and how frequently?

- If our system could compare your energy consumption with your neighbours and friends on Facebook, what kind of comparisons would you like to see?

4.5.2 Results

14 persons answered the questionnaire. The questionnaire results are used as complements to the development and implementation of the use-cases introduced in the previous section. Some other useful points are suggested in the first question such as:

- Advices of how to reduce the current energy usage can be sent as part of the feedback to consumers.
- The ability to track individual appliances and compare with its usage in the past.
- The ability to show statistical data using graphs over time.

For the second question, the answers are variable in question to share the power consumption on social media. Besides, it is important that the frequency is not too high to avoid the superfluous posts on social media. Another valid point not to share the results too often is that not revealing for others when the consumers are at home or not regarding to how much energy data they are using.
With the third question, the answers show that people are interested in comparing their energy usage with friends and neighbours such as:

- Total usage in a household.
- Usage of individual devices.
- Energy usage per square meter.

5

Evaluation

The previous chapter has shown real-world use cases can be implemented in BCStream. Depending on consumers' interest, BCStream can provide different feedback using different continuous queries. In this Chapter we present the results from an evaluation of BCStream. The use-cases, introduced in the previous chapter (see chapter 4), have been implemented as continuous queries, composed by standard operators, and run in BCStream in order to evaluate the system's performance focusing on its throughput (tuples/second) and processing latency (ms). First, we introduce the specification of the hardware (see Section 5.1), followed by a discussion about the input data that we use in the evaluation (see Section 5.2). We perform two different kinds of experiments (see Section 5.3). The first experiment demonstrates how BCStream performs with an off-the-shelf machine and with the second experiment, we evaluate the system's scalability by using different number of the test machine's processor-cores. For every experiment, the use-cases were executed ten times in ten minutes each time. Finally, we present the test results as well as the conclusion.

5.1 Setup

Table 5.1 shows the hardware and software specification for the test machine used for the system evaluation.

OS	Ubuntu 12.04 64bit
Processor	Intel(R) Core(TM) i7 CPU 950 @ 3.07 GHz
CPU cores	4
Memory	6 Gb

Table 5.1: Specification of the test machine.

5.2 Input data

The key to evaluate a system's performance starts with good input data sets. Since the proposed system processes real-time streams of energy usage data continuously, it is crucial to evaluate it with some accurate and realistic energy consumption data sets which are available to public on Tracebase[4]. Tracebase uses Plugwise system [31] to collect real power usage of an attached consumer. Furthermore a polling application is created to request the wattage measured by each of the deployed Plugwise devices. The input data, used for evaluation of BCStream, is obtained from Tracebase and it includes power traces of ten electrical appliances. The collected energy readings are stored in Comma-separated value (CSV) files and each file is generated for each deployed device. Each entry in the CSV files contains the date and time of its collection, followed by the power consumption readings averaged over time durations of one and eight seconds, respectively. An excerpt of an entry is as follow:

28/12/2011 00:00:01;43;43
28/12/2011 00:00:02;45;43
28/12/2011 00:00:04;43;43

Table 5.2: Electrical appliance power traces. The table shows how the energy data retrieved from Tracebase is formatted.

5.2.1 Generate an input data stream using real energy readings.

Since the real energy consumption data, obtained from TraceBase [4], illustrates only the power usage of a customer in one single day, we modify the data to create a data stream in order to evaluate BCStream's performance. We start out with reading the real energy readings stored in Csv files into memory then a Spout (see Section 2.3.5) reads each entry from memory, modifies the entry's date, energy consumption and adds a customer id. Finally, the Spout emits each entry as a tuple to downstream Bolts. The modification is necessary in order to illustrate different power usage of different consumers. By running this process over and over again, we are able to provide BCStream with a real-time continuous input data stream starting with real energy readings and the evaluation becomes more representative of a stream observable in the given system model.

5.3 Test results and discussion

The main objective of the evaluation is to measure the proposed system's performance with Apache Storm as its SPE (see Section 2.3.4), focusing on throughput (tuples/second) and processing latency (ms). Each tuple is 37 bytes long. Two different experiments have been conducted. The first experiment demonstrates how BCStream performs with an off-the-shelf machine and with the second experiment, we want to evaluate the system's scalability by using different number of the test machine's processor-cores. For every experiment, the use-cases are executed ten times for ten minutes each time. As the proposed system's throughput and its processing latency are of highest interest, the use-cases are modified and some functions that are out of scope for the tests are removed, for instance posting notifications to Facebook is exchanged to file-logging. The latency is measured from when the tuple is emitted in the Spout until it reaches the final printing Bolt and the system's throughput is obtained by dividing the total number of emitted tuples to 600 seconds which is the length of each test.

Additionally, in certain applications, there is a need to ensure that every tuple is processed at least once and that results in computational overhead. Apache Storm provides such feature as message guarantee (see Section 2.3.5). Thus, in all experiments, we take into account the execution of the use-cases with and without message guarantee and the results demonstrate if using message guarantee compromises the total performance of BCStream or not. We expect that throughput and latency is better when not relying on message guarantee because of the computational overhead. The results $\big($ throughput (tuples/second) and latencies (ms)$\big)$ are presented with a 95% confidence interval.

5.3.1 Single-node, multi-core evaluation

The first experiment is conducted in order to test how BCStream works in an off-the-shelf test machine (see 5.1). With this experiment, we would like to demonstrate what throughput (tuples/second) can be achieved as well as the processing latencies (ms) using the test machine. This experiment helps the users have an idea of what they receive from BCStream using the test machine in terms of how many tuples the system can process per second and its latency.

Results

Figures 5.1, 5.3, and 5.5 show the results obtained from running the use-cases in our system and the throughput is over what we expected from our proposal which is 100,000 tuples/second. The average throughput is around 151,000 tuples processed every second when in message guarantee mode and 20,000 more per second when not using message guarantee in the first use-case "Daily and weekly peak energy consumption". In the second use-case "Comparing energy usage with neighbours", similar results are achieved. However, in the third use-case "Top five energy consumption appliances", the average throughput is around 140,000 tuples/second with message guarantee enabled and 160,000 tuples/second without supporting message guarantee. The latencies (see Figures 5.2, 5.4 and 5.6) are quite similar in all three use-cases, around 18 ms with message guarantee and 14-16 ms without message guarantee.

As expected, both throughput and processing latency are better when the message guarantee is not enabled (see Section 2.3.5). However, using message guarantee only decreases the throughput with 10% and 2 ms extra as processing latency. It is a trade off that the users have to take in consideration regarding how crucial it is to make sure that all tuples are processed. In addition, these use-cases are not time-critical use-cases since users receive feedback via social media, it is not crucial that the notifications are several seconds late.

Figure 5.1 shows the average throughput for message guarantee and without message guarantee. It clearly shows that without message guarantee feature gives higher throughput, however for these use-cases these latencies (see Figure 5.1) are acceptable. Since the users receive feedback on social media, it is not crucial that the notifications are several seconds late.

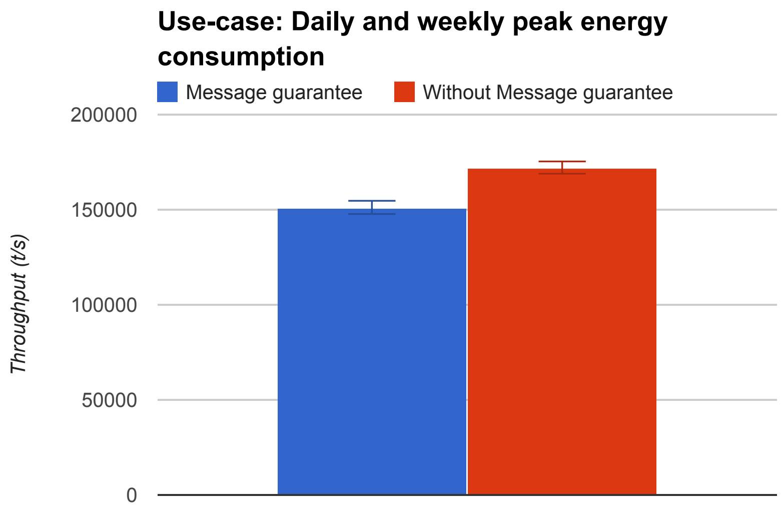


Figure 5.1: System's throughput using the test machine, and 95% confidence interval (Use-case Daily and weekly peak energy consumption).

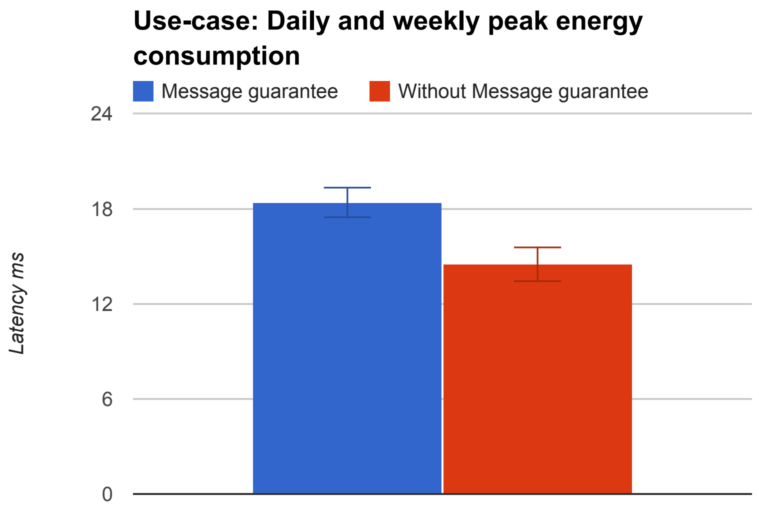


Figure 5.2: Processing latency using the test machine, and 95% confidence interval (Use-case Daily and weekly peak energy consumption).

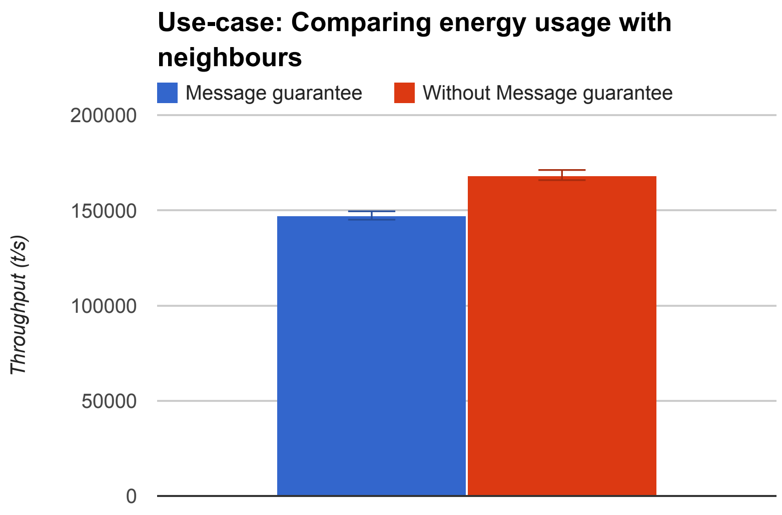


Figure 5.3: System throughput using the test machine, and 95% confidence interval (use-case Comparing energy usage with neighbours).

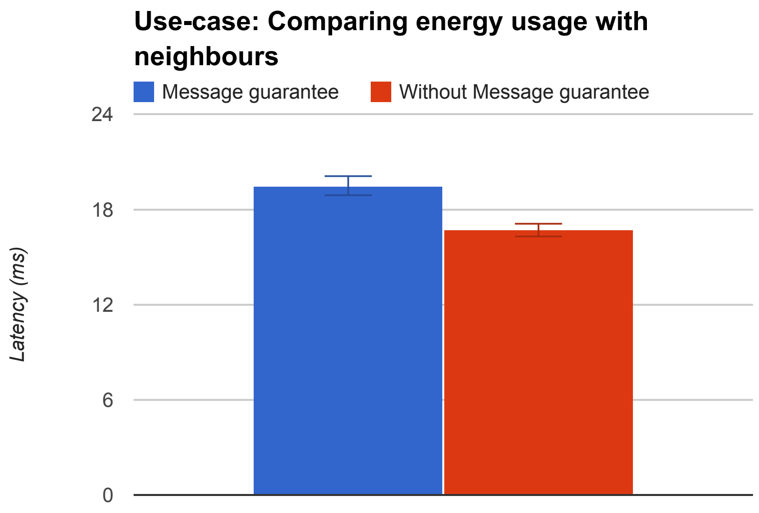


Figure 5.4: Processing latency using the test machine, and 95% confidence interval (Use-case Comparing energy usage with neighbours).

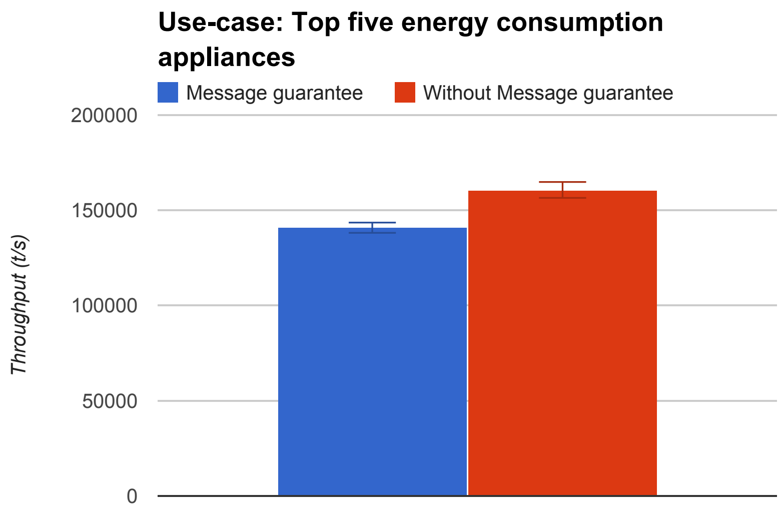


Figure 5.5: System's throughput using the test machine, and 95% confidence interval (Use-case: Top five energy consumption appliances).

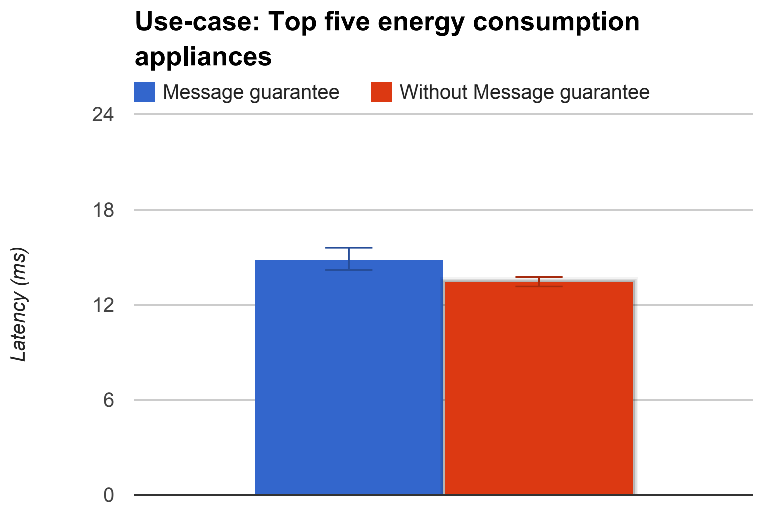


Figure 5.6: Processing latency using the test machine, and 95% confidence interval (Use-case: Top five energy consumption appliances).

5.3.2 Scalability experiment with different numbers of processor-cores

In this experiment, we are evaluating BCStream's scalability utilizing different number of processor cores. This is achieved by controlling the number of utilized processor-cores of the test machine (see 5.1). Intuitively, the more hardware is used in the test-runs, the better throughput is yielded. We start the experiment by allowing the test machine to run one use-case at a time with only one dedicated processor-core. Then we increase the number of processor-cores dedicated for execution of a use-case. The main objective of this experiment is to determine if the proposed system's throughput and processing latency are affected if the system scales.

Results

For all three use-cases, we can see the same results that both throughput and latency are affected by controlling the number of processor-cores of the test machine (see Figure 5.7, 5.8, 5.9, 5.10, 5.11, 5.12). First, as expected the proposed system yields better throughput with more processor-cores utilized in the test machine (see Figures 5.7, 5.9, 5.11). However, it appears to have a small decreasing in throughput when using four processor-cores. It is because of the test machine has four physical processor-cores and therefore BCStream must share processor-cores with other processes running in the computer which causes some overhead and decreases the throughput. Second, the latency decreases significantly while the number of cores used increases (see Figures 5.8, 5.10, 5.12) for executing a use-case. The reason is because the Garbage Collection (GC) has to run more often to free memory as there is only one processor-core dedicated for the system. When the GC is activated other processes must wait until the GC is done therefore the latency is much higher in the case with only one processor-core. Finally, similar to the previous experiment, both throughput and latency are worse when message guarantee is enabled. When one core is utilized the throughput with and without message guarantee are almost the same. However, when several cores are utilized, the average value of the system's throughput increases faster in the case without message guarantee.

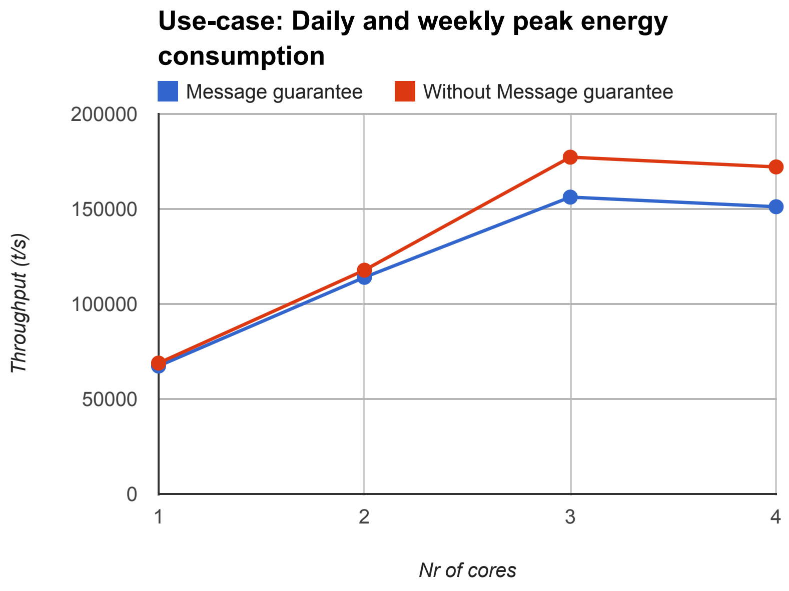


Figure 5.7: System's throughput with different number of processor-cores used (Use-case Daily and weekly peak energy consumption).

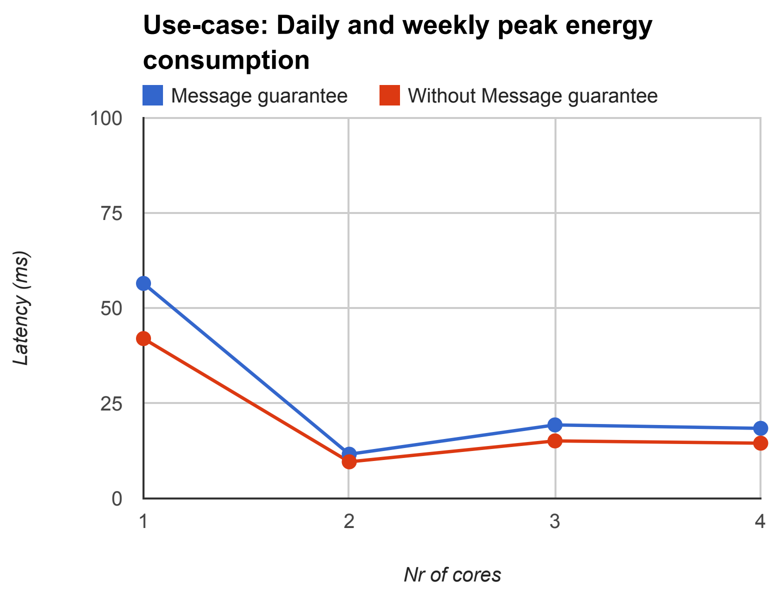


Figure 5.8: Processing latency with different number of processor-cores used (Use-case Daily and weekly peak energy consumption).

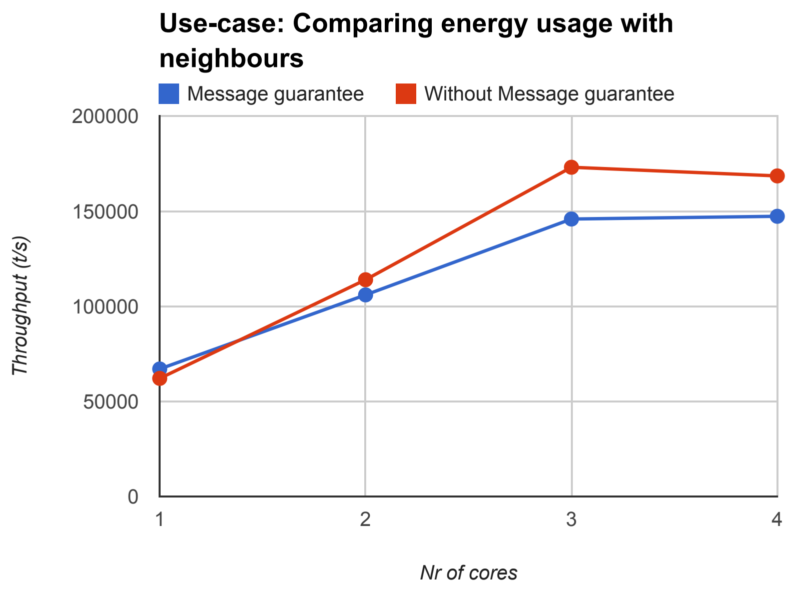


Figure 5.9: System's throughput with different number of processor-cores used (use-case Comparing energy usage with neighbours).

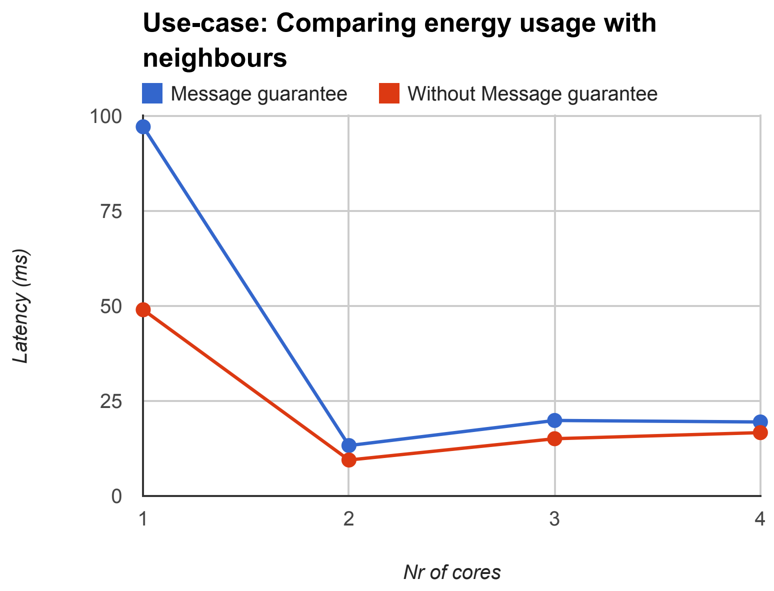


Figure 5.10: Processing latency with different number of processor-cores used (Use-case Comparing energy usage with neighbours).

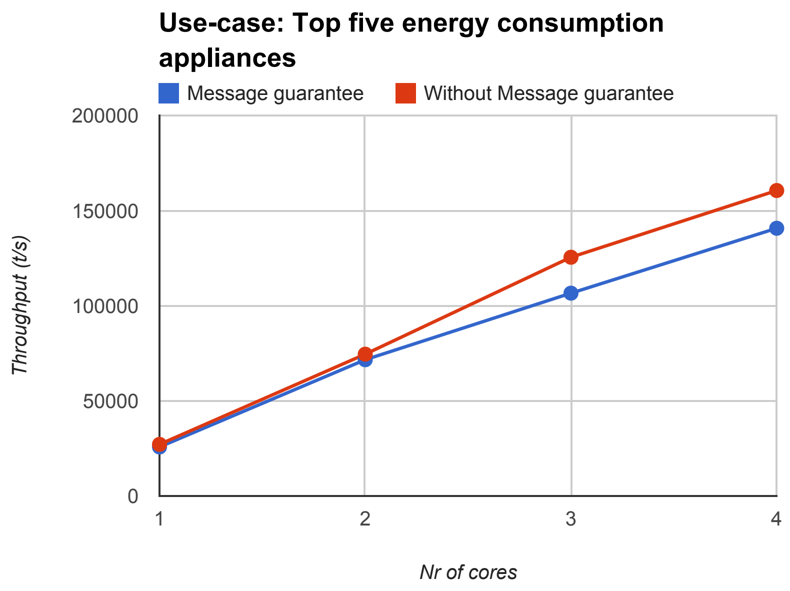


Figure 5.11: System's throughput with different number of processor-cores used (Use-case: Top five energy consumption appliances).

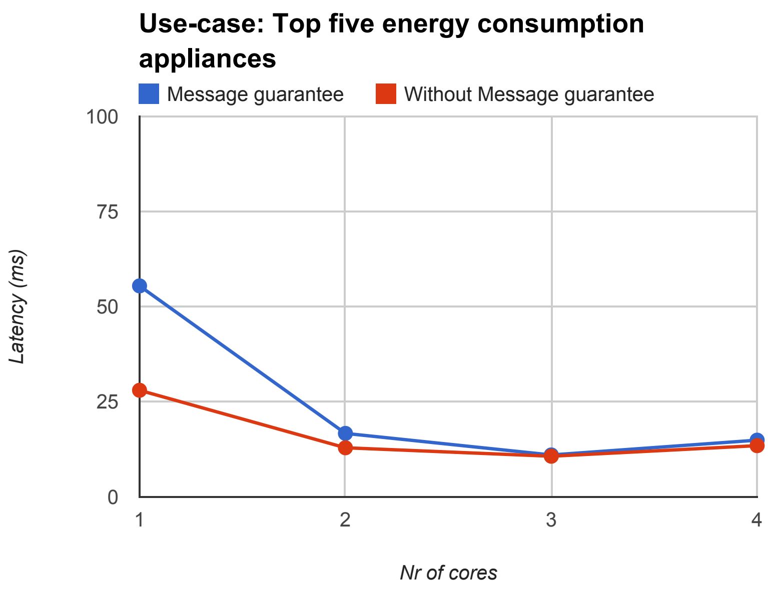


Figure 5.12: Processing latency with different number of processor-cores used (Use-case: Top five energy consumption appliances).

6

Related work

In this Section we presents projects that are related to to our work. The first subsection's (see Section 6.1) focus is mainly on research in the area of smart grids and Advanced Metering Infrastructures (AMIs). This is followed by studies related to social media (see Section 6.2). Furthermore in the third sub section we present work on energy consumption and how instant feedback can be used to motivate people to conserve more energy (see Section 6.3). In the forth sub section we bring up how social media is used to influence people into acting in similar ways (see Section 6.4). Finally, we present existing technologies to process Big Data and some of the available Stream Processing Engines (see Section 6.5).

6.1 Research in smart grid and Advanced Metering Infrastructures

There are a lot of distinct research directions related to smart grids and AMIs such as forecasting of future electricity usage, information security, energy consumption scheduling in smart grids, etc. In this section we introduce studies related to smart grids and AMIs.

With all data being transferred between consumers and suppliers one aspect that must be taken into consideration is the information security. Anthony et al. [32] discuss the security around smart grids and describe key technologies to use, which includes public key infrastructures and trusted computing. Another survey [33] describs how you can divide smart grids into 3 subsystem: Smart infrastructure system, Smart management system and Smart protection system. The last one is the sub system taking care of reliability analysis, failure protection, and security.

One advantage of AMI is the ability to Demand Side Management (DSM). The ability to monitor and control the client devices (Load management) is one of the key functions of AMI. Some customers with smart meters can program their smart appliances so that the devices are utilized when energy is cheaper, and turned off when it is more expensive. If clients on a massive scale run their devices at a time when energy is cheaper, then it would affect the bulk electric grid. Utilities often uses AMIs, especially DSM systems to monitor and control the consumption at the costumers side. These DSM strategies often focus on the two-way interaction between suppliers and customers, however [34] proposes a technique with an incentive-based energy consumption scheduling algorithm which also is autonomous. The work is based on the interaction between the users that only requires a low amount of messages to be exchanged between the costumers. With simulations the show that their technique can reduce both cost and daily user energy consumption, furthermore spread out the usage of energy to lower the consumption peaks.

Forecasting of future power usage is an important task to provide intelligence to the smart gird. Accurate forecasting enables an utility provider to plan the resources and also to take actions to control the balance the supply and the demand of electricity. In [35], a data mining scheme to forecast the peak load of a particular consumer entity in the smart grid for a future time unit is proposed. The experimental results show that the method is able to provide 98.4–98.7% of average accuracy. Moreover, it is also computationally efficient and can potentially be used for large scale load forecasting applications.

6.2 Research in Social media

Social Media is the new way with which we share information, projects or images that we like with the world. People are spending more and more time online, and a large part of this time is spent on social media platforms such as Facebook, Twitter and YouTube. Therefore social media is a great tool for sharing information. In BCStream, insights of energy usage are published on social media in order to raise the energy consumption awareness among consumers. However, social media is a powerful communication tool with great potentials to influence millions of people. Many research have been done to review the various impact of social media in our society. In this section we introduce studies related to social media.

In [36], Clay discuss the political power of social media. He claims that the discussion of the political impact of social media has focused on the power of mass protests to topple governments. In fact, social media's real potential lies in supporting civil society and the public sphere, which produces change over years and decades, not weeks or months.

Yuki Sampei et al. [37] analysed Japanese newspaper coverage of global warming from January 1998 to July 2007. They also conducted a monthly public opinion survey from July 2005 and analysed the changes in public concern about global warming issues and environmental issues in general. They found that coverage of global warming have an immediate but short-term influence on public concern and in order to achieve more effective communication of climate change, strategies aimed at maintaining mass-media coverage of global warming are required. Their study revealed that mass-media coverage of global warming increased slightly overall before January 2007 and then increased dramatically from January 2007. The increasing Japanese media coverage of global warming issues was driven largely by international events involving the USA.

As mentioned earlier, social media emerges as a new medium for providing new sources of information and rapid information. It can be used as a political power [36], or to enhance effective communication of climate change [37]. It is also an information source and contains huge amount of updates and useful information related to real-world events, for instance Twitter's tweets or Facebook updates. In [3], Yin et al. introduces a system that utilizes social media to enhance emergency situation awareness. The proposed system uses natural language processing and data mining techniques to extract situation awareness information from Twitter messages generated during various disasters and crises.

6.3 Energy consumption monitoring & energy saving

The main function of BCStream is to provide its users the opportunity to monitor their energy usage on the fly and receiving notifications through social media. With such insights, the users are able to raise their energy awareness and receive enough information they need to make informed, proactive choices to improve their energy saving. In this section we introduce several studies where they are showing that energy usage monitoring has a positive effect on the awareness of their energy usage and contributes to energy conservation.

When considering the effect of energy monitoring on energy saving, several studies have been undertaken. Matsukawa's study [2] indicates that power usage monitoring contributes to energy conservation. Bakker et al [1] present a 15 months study from 2008 which aimed to investigate if the participants managed to sustain their initial electricity savings over a long period of time (more than four months). The study consists of two phases, an initial trial of four months and a follow-up period of eleven months in order to determine whether the participants can save energy with the help of the monitors under the trial phase and then if the participants can keep up the energy saving trend during the follow-up period. Energy monitors were given to 304 participants and each energy monitor consisted of a sensor, a sending unit, and a display. The sensor and sending unit were attached to the electronic meter furthermore the display received radio signals from the sending unit and showed the power consumption in watts (W) in real-time. The system was also able to show statistics of consumption for the past 24 hours. The results from this study indicates that during the initial trial (first four months), the average savings achieved for all participants were 7.8%. In addition, a group of highly motivated people managed to reach an initial electricity saving of 16.7%. The household with the best result reached a 42.6% lower energy consumption during the pilot and 30.4% during the follow-up phase from the original consumption. This was achieved by strictly recording the electricity meter data twice a day so they could recognize their energy usage pattern and undergo actions in order to decrease their consumption.

A similar study [5] in Japan shows how collecting and presenting energy consumption in nearly real-time manner has a positive effect on the awareness of their energy usage. The energy consumption was registered by a smart meter and then sent to a backend system. The users could than easily access the data using a web browser, where data was presented in hourly, daily, and weekly graphs. The studies demonstrate that energy consumption monitoring results in energy saving but to sustain the initial saving over long time is indeed challenging. However, BCStream emerges as a possible approach for the challenge to maintain its users' interest by providing them not only the power usage monitoring but also the interaction with social media continuously over time. By also offering the consumers more advanced feedback such as comparing consumption with neighbours and find out specific devices consumption, BCStream has advantages in features over the studies recently described.

In 2006, Sarah Darby [38] performed a review of the literature on metering, how the feedback is presented and how it influences the behaviour of the householders. Even though most of the reviewed studies show some kind of improvements in conserving energy with the use of feedback, Darby states that one problem is to keep the conserving persistent. She implies that for it to happen, householders have to develop new habits and proposes a method, where feedback, combined with energy saving advices, should be actualized to ensure that users accomplish the new habits. Her conclusions is that feedback is necessary for energy conservation, however people need help interpret the feedback and guidance on what actions to take. Our system addresses this by integrating energy consumption with social media, consumers can then interact with each other and give helpful hints on what actions could be done to lower the consumption.

In 1996, an experiment [39] was performed with two teams belonging to the same company. The experiment was to see if they could change the energy wasting behaviour for the two groups. Both teams got feedback about their energy consumption, in addition one of the teams also got feedback about how well their competitors were doing. The result from this small experiment shows that both teams managed to save energy and furthermore the second team, the one with comparative feedback, had much more improved conservation results. This was taken into account when designing the use-cases, especially the one where the consumers can compare with their neighbours (Section 4.3).

Chosen as the "Best Idea for the Millennial" in the GE Ecomagination Challenge, start-up Welectricity [40] utilizes data when users update information from their monthly energy bills into the site's graphing tool. Welectricity then creates a readout of consumption habits matched against different types of appliances found in the household, generates stats and charts on energy usage trends, and offers suggestions on how to reduce the power usage. Welectricity also makes it easy for users to include their friends in the process through their online portal. However their model is based on that the users manually gives information about their energy consumption. Our solution eliminates this step and instead enables the data sampling in a more automatized manner.

Finally, another study [41] on energy conservation presents results indicating that one important and promising factor when motivating people to conserve energy is to combine feedback with goal-setting on a personal level, which can be achieved by simply applying a query to notice the users their goals and achievement so far on social medias.

6.4 Energy competitions & social awareness

BCStream provide its users the opportunity to trigger competitions to save energy and raises social awareness by publishing the energy consumption on social media such as Facebook and furthermore users can compare their energy usage with their friends and family. In this section we introduce projects which show that social media influences people's behaviors and actions. In addition, how social games help people to conserve energy by competing with others.

Jeroen Stragier et al.[42] have shown that publishing training results on social media gets positive impact on others. A similar study conducted by Step Foster et al. [43] was performed using a step monitoring device to motivate people to more physical activities. The objective is to integrate with Facebook to compete with others by counting steps taken and registered by the step monitoring device. The results from the study shows that the integration with Facebook which enables the possibilities to view others score, make comments and compare with others' results, tend to motivate the users to engage in more physical activities. The studies have shown that incorporating energy usage into social media to gain social influence and motivate people to conserve energy is indeed possible.

Gustafsson et al. [44] however constructed a game, called PowerExplorer, based on real-time consumption data for instant feedback. The goal of Gustafsson work was to contribute to the discussion on how to design a pervasive learning game, with intention to sustain behaviour changes and learning, within the field of energy efficiency. They recognize a clear problem with these types of games, which is to successfully come up with a long-term change in behavior regarding energy consumption saving. Their solution to the problem lies in the usage of real-time data, so that the users can relate their device usage to energy consumption directly. It is also easier to support faster rewarding if the data is collected in real time which also can enhance the long term change in behaviour. As a result from their field test, they observed 16% energy saving during the game period which lasted for 7 days. As a side effect they also saw how the contestants changes their attitude in a positive way regarding the promotion of energy savings to others.

One also has to remember that the economic gain with lowering the energy consumption might be a contributing factor, not just the fact that it is better for the environment.

6.5 Related technologies to process Big Data

BCStream utilizes Data stream processing paradigm to capture and analyze huge volume of energy consumption data. However, Stream Processing is not the only approach to process Big Data and, therefore, in this section we present the Batch Processing model.

Two of the most wellknown approaches to process Big Data include batch processing and stream processing and the main distinction is that in batch processing, there is a finite amount of data sets collected for a job, whereas in stream processing, the data streams are assumed to be unbounded. Jobs in batch processing produce results only when all computation is done and any changes in the data require reprocessing of the batch job. Batch processing platforms include MapReduce [45] developed by Google, Dryad [46] developed by Microsoft. Apache Hadoop [47] is a wellknown batch processing software built on top of MapReduce framework. There are also some hybrid systems supporting both batch and stream processing such as MapReduce Online [48], Twitter's Summingbird [49] which allows Hadoop working with Storm [24], and Yahoo's Storm Yarn [50]. Depending on the size of the input data sets, and the computational power of the system performing the batch jobs, this may delay users from gaining important insights. Stream processing, on the other hand, enable users to analyze, correlate data and produce incremental results as the data arrives from real-time sources. Batch processing has several benefits such as flexible time of job processing or maximizing the utilization of system's computational power but it is not optimized for the computational latency, continuous feedback, etc., which are the main benefits gained from stream processing applications.

BCStream utilizes Apche Storm as its SPE. However, there exists many other SPEs, from pioneers to commercial ones, are Aurora [51], Esper [52], Amazons Kenisis [53], Microsoft's StreamInsight [19] and IBM InfoSphere Streams[54]. A special case of stream processing is Complex Event Processing (CEP). CEP systems classify tuples in input streams as raw events. A CEP system matches patterns against sequences of events as they arrives in order to generate composite event for each match. Existing CEP systems include NiagaraCQ [55], Esper [52], TIBCO BusinessEvents [56].

7

Discussion

In this chapter, we summarize the conclusions obtained during the development and evaluation of BCStream. Furthermore, we present what could be done in the future in terms of further development of BCStream.

7.1 Conclusion

We present BCStream, a data streaming based application, which utilizes energy consumption data on the fly and produces feedback to users on social media in order to raise their energy usage awareness. BCStream utilizes Apache Storm as its Stream Processing Engine (SPE) to process large amount of continuous data in near real-time. Three sample use-cases are implemented and used to evaluate the system performance in terms of throughput (energy consumption readings per second) and latency (millisecond). The evaluation shows that BCStream can handle up to 170,000 energy consumption readings per second with a latency as low as 14 milliseconds. Besides, the scalability experiment results in at least 60% throughput improvement with every added processor core. The processing latency also improves significantly (decreasing from 80 ms to 20 ms) using multi-processor cores. Apache Storm's feature message guarantee results in additional overhead. It is a trade off that the users need to take into considerations regarding how crucial it is to ensure that all data packets are processed at least once. Moreover, the conducted questionnaire shows that people tend to be interested in getting a better overview of the energy consumption and compare their consumption with friends and neighbors.

A major problem when interacting with social media is to keep the news flow at a moderate level, i.e, not to flood the application with updates. According to [57] an average Facebook user produces 90 pieces of content each month. By letting BCStream interact with Facebook on a weekly basis (around 4 posts a month) the flow of the average user only increases approximately 5%.

7.2 Future work

BCStream is implemented as a prototype in order to evaluate data streaming as a potential way to process and gain valuable insight in energy consumption. Since it is a prototype, it excludes some areas that are not fully explored. For instance, the integration with the AMIs and how energy readings can be sent directly from AMIs to BCStream could be further investigated. Also the questionnaire is conducted in a small group of Chalmers students, which might not reproduce the common users' interest. So a full scale examination of what people think of collecting and sharing energy consumption data could be done in the future. Furthermore, even though the selected SPE delivers over expected performance, this project only evaluate one SPE and future work could include a comparison between several SPEs where performance and functionality are considered. The evaluation is conducted using one single server and the received results are very promising hence another setup with a cluster of servers can be utilized to investigate the performance of BCStream as well as Apache Storm as a distributed stream processing engine.

–

Bibliography

[1] S. S. van Dam, C. A. Bakker, J. D. M. van Hal, Home energy monitors: impact over the medium-term, Building Research amp; Information 38 (5) (2010) 458–469.
URL `http://dx.doi.org/10.1080/09613218.2010.494832`

[2] I. Matsukawa, The effects of information on residential demand for electricity., Energy Journal 25 (1) (2004) 1 – 17.
URL `http://proxy.lib.chalmers.se/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=buh&AN=12000471&site=ehost-live&scope=site`

[3] J. Yin, A. Lampert, M. Cameron, B. Robinson, R. Power, Using social media to enhance emergency situation awareness, IEEE Intelligent Systems 27 (6) (2012) 52–59.

[4] Tracebase.
URL `https://www.tracebase.org`

[5] K. Matsui, H. Ochiai, Y. Yamagata, Feedback on electricity usage for home energy management: A social experiment in a local village of cold region, Applied Energy 120 (0) (2014) 159 – 168.

[6] N. B. Ellison, et al., Social network sites: Definition, history, and scholarship, Journal of Computer-Mediated Communication 13 (1) (2007) 210–230.

[7] J. Tang, J. Sun, C. Wang, Z. Yang, Social influence analysis in large-scale networks, in: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '09, ACM, New York, NY, USA, 2009, pp. 807–816.
URL `http://doi.acm.org/10.1145/1557019.1557108`

[8] A. Goyal, F. Bonchi, L. V. Lakshmanan, Learning influence probabilities in social networks, in: Proceedings of the Third ACM International Conference on Web

Search and Data Mining, WSDM '10, ACM, New York, NY, USA, 2010, pp. 241–250.
URL `http://doi.acm.org/10.1145/1718487.1718518`

[9] A. Anagnostopoulos, R. Kumar, M. Mahdian, Influence and correlation in social networks, in: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '08, ACM, New York, NY, USA, 2008, pp. 7–15.
URL `http://doi.acm.org/10.1145/1401890.1401897`

[10] H. Farhangi, The path of the smart grid, Power and Energy Magazine, IEEE 8 (1) (2010) 18–28.

[11] SmartGrid.gov.
URL `https://smartgrid.gov`

[12] B. Hoang, Smart grids, IEEE Emerging Technology portal 2012.
URL `http://www.ieee.org/about/technologies/emerging/emerging_tech_smart_grids.pdf`

[13] It's your smartgrid by GE.
URL `http://www.itsyoursmartgrid.com`

[14] Advanced Metering Infrastructure - National Energy Technology Laboratory (2008).
URL `http://www.netl.doe.gov/File%20Library/research/energy%20efficiency/smart%20grid/whitepapers/AMI-White-paper-final-021108--2--APPROVED_2008_02_12.pdf`

[15] GOV.UK.
URL `https://www.gov.uk/smart-meters-how-they-work`

[16] A. Kumar, S4ning smart grid architectural thinking using stream computing.

[17] S4 - http://incubator.apache.org/s4.
URL `http://incubator.apache.org/s4`

[18] S. Z. Sbz, S. Zdonik, M. Stonebraker, M. Cherniack, U. C. Etintemel, M. Balazinska, H. Balakrishnan, The aurora and medusa projects, IEEE Data Engineering Bulletin 26.

[19] Microsoft StreamInsight.
URL `http://msdn.microsoft.com/en-us/sqlserver/ee476990.aspx`

[20] D. N. D. Terry, D. Goldberg, B. Oki, Continuous queries over append-only databases, Proc. of the 1992 ACM SIGMOD Intl. Conf. on Management of Data, pages 321-330.

[21] V. Gulisano, R. Jimenez-Peris, M. Patino-Martinez, C. Soriente, P. Valduriez, Streamcloud: An elastic and scalable data streaming system, Parallel and Distributed Systems, IEEE Transactions on 23 (12) (2012) 2351–2365.

[22] K. Patroumpas, T. Sellis, Window specification over data streams.

[23] R. Peris, M. Martinez, Parallel processing of continuous queries on data streams, uS Patent App. 13/112,628 (Dec. 22 2011).
URL `http://www.google.com/patents/US20110314019`

[24] Apache Storm.
URL `http://storm.incubator.apache.org/documentation/Home.html/`

[25] Ibm Hadoop.
URL `http://www-01.ibm.com/software/data/infosphere/hadoop/`

[26] Hitatich - Home Energy Management System (HEMS).
URL `http://www.hitachi.com/products/smartcity/solution/hems/index.html`

[27] RabbitMQ- https://www.rabbitmq.com/.
URL `https://www.rabbitmq.com/`

[28] Apache Kafka - http://kafka.apache.org/.
URL `http://kafka.apache.org/`

[29] Foursquare.
URL `https://foursquare.com/`

[30] List of Foursquare badegs.
URL `http://www.4squarebadges.com/foursquare-badge-list/active-badges/index.html`

[31] Tracebase - Smart Home & Smart Building.
URL `http://www.plugwise.com/`

[32] A. Metke, R. Ekl, Security technology for smart grid networks, Smart Grid, IEEE Transactions on 1 (1) (2010) 99–107.

[33] X. Fang, S. Misra, G. Xue, D. Yang, Smart grid x2014; the new and improved power grid: A survey, Communications Surveys Tutorials, IEEE 14 (4) (2012) 944–980.

[34] A.-H. Mohsenian-Rad, V. Wong, J. Jatskevich, R. Schober, A. Leon-Garcia, Autonomous demand-side management based on game-theoretic energy consumption scheduling for the future smart grid, Smart Grid, IEEE Transactions on 1 (3) (2010) 320–331.

[35] Z. Aung, M. Toukhy, J. Williams, A. Sanchez, S. Herrero, Towards accurate electricity load forecasting in smart grids, in: DBKDA 2012, The Fourth International Conference on Advances in Databases, Knowledge, and Data Applications, 2012, pp. 51–57.

[36] C. Shirky, Political power of social media-technology, the public sphere sphere, and political change, the, Foreign Aff. 90 (2011) 28.

[37] Y. Sampei, M. Aoyagi-Usui, Mass-media coverage, its influence on public awareness of climate-change issues, and implications for japan's national campaign to reduce greenhouse gas emissions, Global Environmental Change 19 (2) (2009) 203–212.

[38] S. Darby, The effectiveness of feedback on energy consumption, A Review for DEFRA of the Literature on Metering, Billing and direct Displays 486 (2006) 2006.

[39] F. W. Siero, A. B. Bakker, G. B. Dekker, M. T. van den Burg, Changing organizational energy consumption behaviour through comparative feedback, Journal of Environmental Psychology 16 (3) (1996) 235–246.

[40] Welectricity.
URL `http://welectricity.com/home`

[41] J. H. v. Houwelingen, W. F. v. Raaij, The effect of goal-setting and daily electronic feedback on in-home energy use, Journal of Consumer Research 16 (1) (1989) pp. 98–105.
URL `http://www.jstor.org/stable/2489305`

[42] J. Stragier, P. Mechant, Mobile fitness apps for promoting physical activity on twitter: the #runkeeper case, in: Etmaal van de communicatiewetenschappen, Proceedings, 2013, p. 8.

[43] D. Foster, C. Linehan, B. Kirman, S. Lawson, G. James, Motivating physical activity at work: using persuasive social media for competitive step counting, in: Proceedings of the 14th International Academic MindTrek Conference: Envisioning Future Media Environments, ACM, 2010, pp. 111–116.

[44] A. Gustafsson, M. Bång, M. Svahn, Power explorer: a casual game style for encouraging long term behavior change among teenagers, in: Proceedings of the International Conference on Advances in Computer Enterntainment Technology, ACM, 2009, pp. 182–189.

[45] J. Dean, S. Ghemawat, Mapreduce: Simplified data processing on large clusters, Commun. ACM 51 (1) (2008) 107–113.
URL `http://doi.acm.org/10.1145/1327452.1327492`

[46] Microsoft Dryad.
URL `https://github.com/MicrosoftResearchSVC/Dryad`

[47] J. Turner, Hadoop: What it is, how it works, and what it can do (2013).
URL `http://strata.oreilly.com/2011/01/what-is-hadoop.html`

[48] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, R. Sears, Mapreduce online., in: NSDI, Vol. 10, 2010, p. 20.

[49] Twitter SummingBird.
URL `https://github.com/twitter/summingbird/wiki`

[50] Yahoo's Storm Yarn - Streaming Hadoop.
URL `https://github.com/yahoo/storm-yarn`

[51] The Aurora Project.
URL `http://cs.brown.edu/research/aurora/`

[52] Esper EsperTech.
URL `http://esper.codehaus.org/`

[53] Amazon S3.
URL `https://aws.amazon.com/kinesis`

[54] InfoSphere Streams.
URL `http://www-03.ibm.com/software/products/en/infosphere-streams`

[55] J. Chen, D. J. DeWitt, F. Tian, Y. Wang, Niagaracq: A scalable continuous query system for internet databases, SIGMOD Rec. 29 (2) (2000) 379–390.
URL `http://doi.acm.org/10.1145/335191.335432`

[56] TIBCO BusinessEvents CEP.
URL `http://www.tibco.com/products/event-processing/complex-event-processing/businessevents/default.jsp`

[57] jeffbullas.com.
URL `http://www.jeffbullas.com/2011/04/28/50-fascinating-facebook-facts-and-figures/`

[58] J. Brody, P. Yager, R. Goldstein, R. Austin, Biotechnology at low reynolds numbers, Biophysical Journal 71 (6) (1996) 3430 – 3441.
URL `http://www.sciencedirect.com/science/article/pii/S0006349596795383`