Parallel Processing of Data from Very Large-Scale Wireless Sensor Networks

Christine Jardak, Janne Riihijärvi, Frank Oldewurtel, and Petri Mähönen
Institute for Networked Systems
RWTH Aachen University
Kackertstrasse 9, D-52072 Aachen, Germany
{cja,jar,fol,pma}@inets.rwth-aachen.de

ABSTRACT

In this paper we explore the problems of storing and reasoning about data collected from very large-scale wireless sensor networks (WSNs). Potential worldwide deployment of WSNs for, e.g., environmental monitoring purposes could yield data in amounts of petabytes each year. Distributed database solutions such as BigTable and Hadoop are capable of dealing with storage of such amounts of data. However, it is far from clear whether the associated MapReduce programming model is suitable for processing of sensor data. This is because typical applications MapReduce is used for, currently are relational in nature, whereas for sensing data one is usually interested in spatial structure of data instead. We show that MapReduce can indeed be used to develop such applications, and also describe in detail a general architecture for service platform for storing and processing of data obtained from massive WSNs.

Categories and Subject Descriptors

D.1.3 [**Programming Techniques**]: Concurrent Programming—*Parallel Programming*

Keywords

MapReduce, Hadoop, parallel processing

1. INTRODUCTION

Wireless Sensor Networks, tiny measurement devices connected by means of wireless communications, have been a topic of intensive research over the past decade. Deployments of Large-scale WSNs (LS-WSNs) have potential to shed light on a number of environmental phenomena, such as progression of climate change, pollution levels, effects of urbanization, and so on. Several initiatives around the world, see e.g. [19] and [13] are actively studying the problems of interconnecting such vast numbers of sensor nodes. Also companies such as Nokia are actively looking into integration of sensing capabilities into mobile phones [24], potentially introducing billions of sensing capable devices into the Internet. Earlier work in the field has mainly focused on the interconnection problem, developing solutions for energy efficient medium access control [14], multihop routing [18] and management of WSNs. However, in addition to the highly interesting technical challenges related to WSNs themselves, their widespread deployment would also require development of solutions for storing and reasoning about the potentially huge amounts of data they would generate. Somewhat surprisingly, this avenue of research has received very little attention thus far.

In this paper we study how suitable existing distributed database (DB) solutions, and especially the MapReduce programming model would be for the basis of such infrastructure for storing and analyzing sensor data. We specifically show through a detailed case study how to implement typical spatial data processing tasks. We focus especially on environmental data in regions in which direct measurements are not available, on MapReduce. We also discuss the general architecture of our system, providing a flexible framework in which different applications for processing of data obtained from WSNs can be implemented on. As a basis we define a DB scheme consisting of a collection of tables for storing individual sensor readings, as well as further information on the networks themselves, such as estimates of accuracies of the sensors used. On top of this basic collection of tables we propose a framework for analyzing and modeling sensor including estimation of various spatial statistics of interest. We also give initial performance results for the implemented algorithms for spatial data processing, demonstrating that using distributed databases and the MapReduce programming model for dealing with vast amounts of data obtained from WSNs is feasible.

The rest of the paper is structured as follows. We discuss the existing work for large-scale WSNs, parallel processing framework, and distributed data storage in Section 2. After that we describe the system architecture in Section 3 focusing on the data model, the processing framework, and the literature behind the estimation of a spatial phenomenon. Section 4 is the main section where we describe the implemented modules for realizing the latter spatio-temporal analysis. We demonstrate the performance evaluation of the system in Section 5. We conclude the discussion with Section 6.

HPDC'10, June 20–25, 2010, Chicago, Illinois, USA.

2. RELATED WORK

The emergence of LS-WSNs means that applications and data produced scale beyond the capabilities of a single server for storing, processing and managing massive amounts of sensor data. We discuss in this section the background work of LS-WSNs, the available parallel processing frameworks, and concentrate on some of the distributed commercial databases for their reliability and availability.

2.1 Large-scale WSNs

Environmental monitoring has been pointed out by several authors as one of the promising, high-value applications for LS-WSNs [24]. Both urban and planetary scale rural applications have been envisioned. A typical scenario assumes that a large number of sensor platforms are mounted on pedestrians or vehicles. Each platform collects its sensor readings and reports them back to a powerful end user machine where data is stored and analyzed measuring human exposure to pollutants. While this is the common network architecture for most of the LS-WSNs, the key difference is the used platforms. The Nokia research group [24] benefited from the facts that mobile phones are widely used and considered equipping them with different types of sensors suitable to applications such as fitness monitoring and mobile environmental sensing. In [2] authors equip people in Cambridge with sensors to monitor the air pollution in the city. The sensor data is communicated to the mobile phone via Bluetooth and further sent via GPRS to an analyze center. Other projects such as Common Sense from Intel [23] developed a new mobile sensing platform on street sweepers.

2.2 Parallel processing framework

MapReduce [16] aims at a good performance taking into account the hardware and software heterogeneity in a cluster. It enables applications, with large processing power requirements to be split into smaller tasks that are executed in parallel on the available machines. Similarly, River [25] targets a good performance despite non-uniformities in the system. However, its performance is attributable to two different features. First, the usage of distributed queues balancing the work across consumers in the system. Second, a graduated declustering mechanism distributing the load of data production among several producers. A different concept of parallel processing is the Bulk-Synchronous Parallel (BSP) model [20], which is considered as a bridge between software and hardware. Similarly to BSP some MPI primitives [26] offer a high level of abstraction facilitating the parallel programming. MapReduce is closer to the developer by offering a defined programming model parallelizing the user programs and a transparent fault-tolerance. Our search for applications on MapReduce led us to [7], a close work to ours where authors develop an R-tree used for spatial access methods. Our system presents a more complete stack of modules touching the construction of a 2d-tree, proceeding to the modeling of the spatial phenomenon, and ending with a spatial prediction application.

2.3 Distributed data storage

Oracle's Real Application Cluster DB [6] cooperates as a single system. It distributes the workload across the cluster by allocating resources to applications and decomposes an application into smaller components. Opposite to [6], the

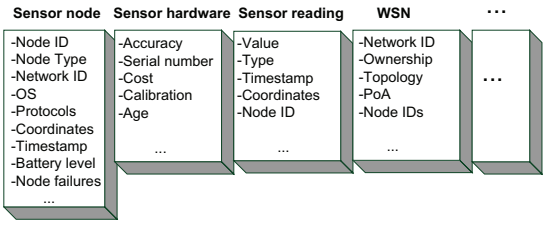

Figure 1: Sensor data tables.

IBM DB2 Parallel Edition [9] and Bigtable [10] are a group of servers that use messaging to exchange data with each other. Tables are split into tablets distributed on the cluster nodes. Both systems employ a hash partitioning strategy on a set of columns to determine the node storing a defined row.

HBase [4] implemented in JAVA, and Hypertable [5] implemented in C++ are two Bigtable-alike systems build on top of Hadoop MapReduce [3] programming model. From the functional standpoint both databases are very similar. The stored data is organized into tables, rows and cells. Each cell in the table is indexed by a row key, column key and a timestamp. The timestamp allows a cell to contain multiple versions of the same data. An iterator-like interface is available for scanning through columns. While Hypertable allows different logical column families to be together physically, HBase allows it for the columns of a single family. On the other hand Hbase supports Bloom filters to speed the access making the performance comparison of both projects difficult. The offered wrappers between both tables and MapReduce [16] encourage the development of novel data processing applications. Moreover, HBase has been put through some fairly significant paces by several companies, which is the major factor behind our decision for using it in this work.

3. SYSTEM ARCHITECTURE

Typically adopted infrastructures for a WSN consider reporting the collected sensor data to backend machines where it is stored and analyzed, as our deployed sensor networks in [8] and [15]. Despite the fact that each WSN is managed by a different organization, networks will not remain isolated. The need of processing sensor data on a scale of countries will arise, resulting into interconnecting the different backend machines in a single cluster. In this paper we consider massive amount of sensor data received by the cluster. We study the adopted data model, and a framework for analyzing and modeling sensor data such as spatial interpolation.

3.1 Data model

The richness of the collected sensor data opens wide ranges of analysis. Thus, we briefly detail some of the interesting data to collect and sort them in different structures further referred to as tables. We consider four primitive tables as depicted in Figure 1: A table for the sensor nodes, a table for the sensor hardware, a table for the sensor readings, and a table for WSNs. Additional data and tables are possible and highly depend on the scope of the analysis to conduct.

The first table maintains data related to the sensor node, i.e., identifiers for the node, node type, and the WSN. It

lists the software running on the node i.e., operating system and protocols. Due to the fact that a node can be mobile, it is highly important to register its geographical location every period of time, i.e. there is a requirement for the coordinate and timestamp fields. Additional fields informing on the battery level and the number of rebooting of the nodes serve for conducting statistics on the status and lifetime of the network. The second table collects the characteristics of the sensor hardware, i.e., accuracy, serial number, and cost. The current status of the hardware is reflected in the calibration field and the age field counting the number of collected reading samples. The third table stores data related to the sensor reading, such as the value and the type, the time and the location at which the reading has been taken, and the ID of the sensor node in charge. The fourth table widens the scope of the sensor data providing general information on the WSNs. The network is identified with a global ID and the coordinates of the owner, we list as well some network criteria such as: The topology of the WSN, the gateway IP address defining the point of access (PoA) to the network, and the pool of the node IDs forming the WSN.

3.2 Processing framework

We aim at providing a complete framework for processing sensor data realized in MapReduce. The targeted package offers the researchers the possibility of executing various parallel processed analyses on massive amount of sensor data. Inspired from the literature of WSNs [21], we sort the proposed analyses into four categories: The acquisitional, aggregate, range, and spatio-temporal analysis. For each category, we introduce examples highlighting their links with the maintained sensor data tables.

The acquisitional analyses are simple and search sensor data of a specific node such as the calibration status of a sensor, number of rebooting, or the battery level. The aggregate analyses are the ones acquiring the average value of a sensor data type from a set of nodes such as the average value of a reading, or of failed nodes. The range analyses target nodes whose information value falls into a predefined range. An expressive example is revealing the trust level of the sensor readings. This results into the need of processing the readings from sensors whose accuracy is bounded by a minimal and maximal value. More complex, but highly research oriented types of analyses are the spatio-temporal ones. Generally, this category follows the principle of a data windowing, i.e. selecting a geographical region. These analyses are applied on the window-contained nodes alone. We refine the scope on three spatial statistics of sensor readings. The first calculating the correlation of a phenomenon between two geographical windows. The second is interesting for the global warming and analyzes the trend of an environmental factor, i.e. temperature. The third example considers a window of nodes for spatial prediction of a phenomenon. The high costs of deploying WSNs in harsh environments providing no infrastructure result into a nonuniform geographical distribution of sensor nodes. The spatial interpolation of a phenomenon in such regions is highly important, and is the main focus of this work.

3.3 Spatial interpolation

Due to space reasons, we treat the particular case of spatial prediction of a phenomenon as an exemplary model for our case study. This analysis requires a function for describing the spatial correlation of an observation. One possibility is the semivariogram which is used to fit a model of the spatial correlation of an observed phenomenon. However, one should make a distinction between the experimental semivariogram that is a visualization of a possible spatial correlation and the semivariogram model that is further used to define the weights for the *kriging* function [22]. That is, finding the best linear unbiased estimation of the random field values between the observed readings.

Assuming n is the total number of nodes in a region, the natural way to interpolate a reading value at a location (s_0) is then realized in three major steps: First, finding the experimental semivariogram of the phenomenon in this region. Second, determining the semivariogram model. Third, defining the weights of the kriging function and then interpolating the unobserved value.

We consider a set of observations $Z(s_i)$, temperature readings in our case, made at the n different locations $s_i \in D$. The semivariogram is half of the expected squared increment of values between locations s_i and s_j,

$$\gamma(s_i, s_j) = \frac{1}{2} E(|Z(s_j) - Z(s_i)|^2). \tag{1}$$

We estimate the semivariogram by employing a binning approach. Let h denote a distance and $N(h)$ the set of pairs (i, j) such that $h - \Delta \leq |s_j - s_i| \leq h + \Delta$. The trivial way of estimating the mean yields the empirical semivariogram

$$\hat{\gamma}(h) = \frac{1}{2\,|N(h)|} \sum_{N(h)} \left(Z(s_j) - Z(s_i)\right)^2. \tag{2}$$

The prediction of an unobserved reading value demands a proper semivariogram model. Thus, a good predictor depends on finding the best fitting model. The common choices for the modeling are the Gaussian model, the Matérn model, and the exponential model which we use

$$\gamma_{exp}(h) = a + b(1 - e^{-h/c}), \tag{3}$$

with the variables $a, b, c \in \mathbb{R}^+$. The optimal values for the variables are the ones minimizing the value of the exponential model.

Having the semivariogram model, we then need to infer the unobserved temperature value at location s_0 from the readings observed at the n known spatial locations $s_1, \cdots, s_n$. We obtain $Z(s_0)$ by applying the kriging, which can be thought as a kind of stochastic version of linear interpolation. The value of $Z(s_0)$ is computed using

$$\hat{Z}(s_0) = \sum_{i=1}^{n} \lambda_i Z(s_i), \tag{4}$$

where the weights λ_i are calculated to minimize the prediction error. For details on the derivation of the weights we refer the reader to [22]. The formula of the weight coefficients $\lambda \equiv (\lambda_1, \cdots, \lambda_n)$ is given by

$$\lambda' = \left(\gamma + 1\frac{(1 - 1'\Gamma^{-1}\gamma)}{1'\Gamma^{-1}1}\right)' \Gamma^{-1}. \tag{5}$$

From the analytical semivariogram in (3), we calculate the matrices $\gamma \equiv (\gamma(s_0 - s_1), \cdots, \gamma(s_0 - s_n))'$ and Γ. The Γ is

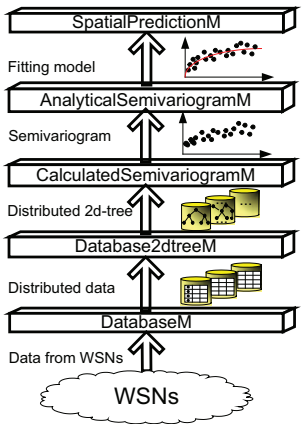

Figure 2: Data computation flow.

the $n \times n$ matrix whose (i, j)th element is $\gamma(s_i - s_j)$. Having 1 as a matrix of n ones, we then substitute the values of γ, Γ, and 1 in (5), calculate the weight coefficients, and predict the temperature value at location s_0 employing (4).

3.4 Enabling techniques

We aim to optimize the computation time of the spatial prediction. We propose structuring a network of nodes into a 2-dimensional tree (2d-tree) minimizing the spatial search time. Moreover, we choose one of several minimization methods for evaluating the variables of the fitting model.

Each sensor data table is stored as a distributed DB on the end user machines. Considering that a sensor node reserves a single field in the DB for storing its geographical coordinates, a spatial search in such a database reads $n \times n$ fields for computing a result. While this is feasible for small size databases, it degrades the performance of systems keeping large size databases. We enable this sort of analyses by structuring the nodes into a 2d-tree. A k-dimensional tree in general is a data structure storing a finite set of points from a k-dimensional space. It was examined in detail in [17], and proved to optimize the search time to $O(k \times n^{(1-(\frac{1}{n}))})$.

The question we answer now is determining the values of the variables a, b, and c, that is finding the best-curve to the given set of points of the experimental semivariogram. In the literature several mathematical methods [22] are described. The least squares fitting method is one of them. The best fit between modeled data and observed data, in its least-squares sense, is an instance of the model for which the sum of squared residuals has its least value. However, this method takes no cognizance of the distributional variation of the generic estimator. For this reason we use the weighted least squares fitting [22] approximated by minimizing

$$\sum_{j=1}^{k} |N(h(j))| \left\{ \frac{\hat{\gamma}(h(j))}{\gamma_{exp}(h(j))} - 1 \right\}^2. \quad (6)$$

4. IMPLEMENTATION DETAILS

Modeling the semivariogram of a phenomenon in a geographical area is implemented via four main modules as depicted in Figure 2: *DatabaseM*, *Database2dtreeM*, *CalculatedSemivariogramM*, and *AnalyticalSemivariogramM*. The spatial prediction using the semivariogram to infer an unobserved temperature value at a specific location is achieved by the single module *SpatialpredictionM*. We describe the implementation details of each module. In case of parallel processed module, we define the input and output values for the map and reduce functions. We highlight as well the mathematical calculations realized in the mapper and the ones realized in the reducer.

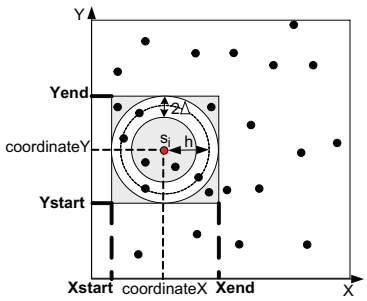

Figure 3: Identifying the node pairs.

4.1 Observation model

Before proceeding with the implementation details of each module, we describe the observation model generating the sensor readings of a phenomenon in a region. The model is implemented in MATLAB assuming the generated phenomenon to be a realization of a random field. That is, a stochastic process defined on a region. Any spatial correlation inherent to most physical phenomena can be taken into account by varying the parameters controlling the statistical structure of the random field. Assuming a fixed region, the coordinates of all nodes are defined by a random point process. The simplest example of such process is the often used Poisson process. Spatially correlated data fields and various random point processes have been studied in our previous works [12] and [11].

4.2 DatabaseM

The DatabaseM module writes the sensor data in an HBase database. It is a MapReduce Java program implementing a mapper. The input value of the map function is the paths of the sensor data files. The module launches several mappers reading the data from the text files and writing them into a specific HBase. The DB called *NodesDB* is characterized by a single column family *cf* and four columns: *coordinateX*,*coordinateY*,*nodeID*, and *temperatureValue*.

4.3 Database2dtreeM

The construction of the 2d-tree and the implementation of the adequate search mechanism are realized in Database2dtreeM. It is a simple Java class having two main functions: A constructor and a search function. The constructor reads the coordinates of the nodes from NodesDB, sorts them into a 2d-tree called *2dtreeDB*. The search function identifies the nodes laying in a predefined geographical region. We show in Figure 3 an exemplar region bounded with the coordinates $[Xstart; Xend]$ and $[Ystart; Yend]$.

4.4 CalculatedSemivariogramM

The CalculatedSemivariogramM is a MapReduce program implementing a mapper and a reducer. Its major job is computing the experimental semivariogram. For a predefined distance h, the mapper searches in the 2dtreeDB all the nodes falling into the correspondent bin, and calculates a portion from the sum of the semivariance. Upon reception of all the partial sums the reducer computes the final value of the semivariance.

The map function takes as an input value the name of the NodesDB database. For a predefined distance h, the map function iterates on the total number of nodes n in the DB performing: A search in the 2dtreeDB database, a filtering process, and a partial calculation of the semivariance. As depicted in Figure 3 For each node located at s_i having a temperature $Z(s_i)$, the search process defines the boundaries of the geographical region and searches in the 2dtreeDB the nodes laying in this region. The located nodes have heterogeneous distance values, and thus, need to be further filtered. The filtering process calculates the distances d from the node located at s_i to all other nodes in the region. Nodes whose distances fall into the correspondent bin of h, verifying $h - \Delta < d < h + \Delta$ form the output set, while others are filtered out. Referring to the figure, the output set consists of the nodes located in the white ring, and the ones belonging to the shadowed regions are filtered out. Finally, the mapper calculates a portion from the sum of the semivariance corresponding to all the node pairs formed from the one located at s_i with all the members of the output set. The output value of the mapper is a string concatenating the value of the sum together with the number of found pairs. The output key of the mapper is the value of distance h.

The reduce function iterates on the strings received from the mapper extracting the sums, and the counters. Then, it computes the estimated semivariance $\hat{\gamma}(h)$. The output value of the reducer is the value of the semivariance, and the output key is the distance h.

4.5 AnalyticalSemivariogramM

The implementation of a non linear conjugate gradient optimizer is realized in the AnalyticalSemivariogramM module. The latter is a simple Java class using the online mathematical functions offered by the apache package [1]. We content to note that the module takes the following input parameters: The sets of h distances, $\hat{\gamma}(h)$, and counters $N(h)$ generated by the previous module and the analytical expression to minimize defined in (6).

4.6 SpatialpredictionM

The SpatialpredictionM module is used for interpolating a phenomenon. It is a MapReduce program implementing a mapper and a reducer. The map function calculates rows from the temperature, γ, and Γ matrices. In its turn, the reduce function constructs the full matrices and interpolates the spatial temperature value.

The map function takes as an input value the name of a database storing spatial locations lacking of sensor nodes. We target to distribute the computation time of the interpolation. Thus, we allow a single mapper to predict the missing temperature values of a predefined set of spatial locations. The mapper iterates on all the nodes in its set, for each location s_0 in the set and s_i in the sensor network it calculates: The $\gamma(s_0 - s_i)$, and the row of the Γ matrix calculated from the location s_i to all other nodes in the network. The output value of the map is a string concatenating the two latter parameters together with the temperature value at s_i. The output key of the map is the location s_0.

Upon receiving the key and string values from the mapper, the reduce function starts building three matrices: First, it combines the n values of $\gamma(s_0 - s_i)$ and forms the γ matrix. Second, it combines the n different received rows and constructs the full Γ matrix. Third, it stores the n temperature values $Z(s_i)$ in a matrix. From the first two matrices, the reducer computes the matrix of weight coefficients. Using the latter matrix and the temperature matrix, the reducer interpolates the temperature at s_0. The output keys of the reducer are the different spatial locations s_0 and its output values are the interpolated temperature readings.

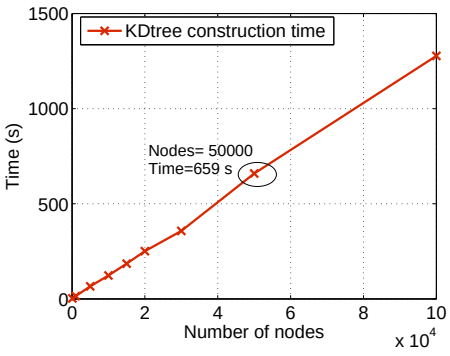

Figure 4: The time consumed for constructing a 2d-tree function of the number of nodes.

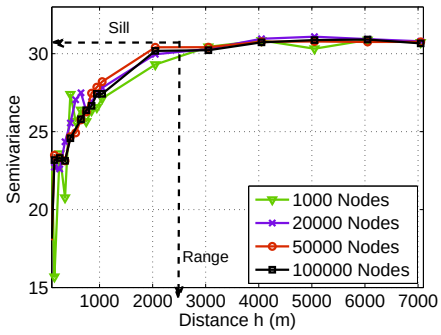

Figure 5: The semivariogram function of the distance reaching a sill of 31 and a range of 2500 m.

5. SYSTEM EVALUATION

In this section we evaluate each of the constructed modules in order to understand experimental limitations and possibilities of LS-WSNs data processing. We first detail the scenario we envisage highlighting the settings of the modules, the special configuration parameters of Hadoop, and outline

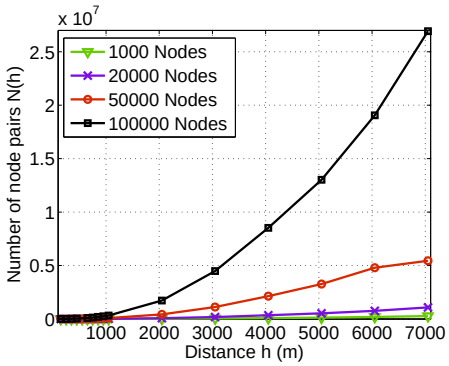

Figure 6: Number of node pairs $N(h)$ separated h meters one from another.

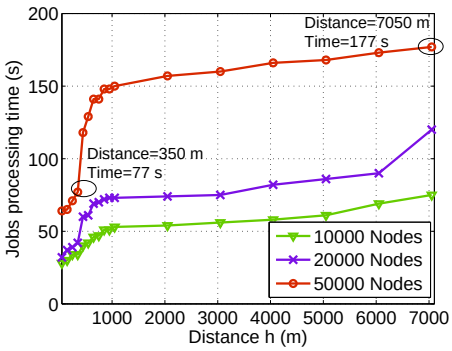

Figure 8: The jobs processing times function of the distance h.

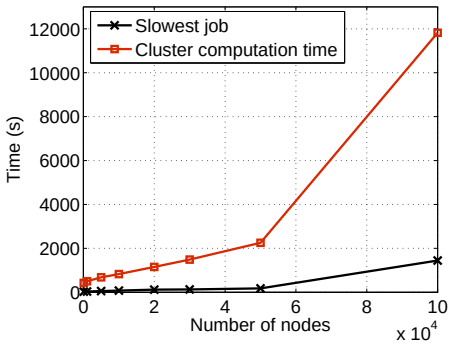

Figure 7: The cluster computation time and the slowest job time for different network sizes.

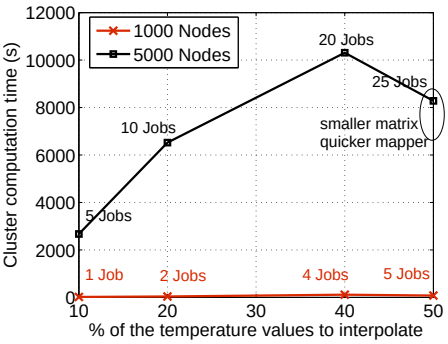

Figure 9: The computation time of the whole cluster function of the percentage of temperature readings to interpolate.

the cluster setup. For modeling the semivariogram of a phenomenon we emphasize the performance of the Database-2dtreeM and the CalculatedSemivariogramM. In the spatial prediction we evaluate the performance of the exemplar SpatialpredictionM.

5.1 Scenario and prototyping plot

Our scenario considers a portion of the massive sensor data stored in a distributed DB. We choose to analyze the nodes located in a region of 29000×29000 square meters, a size of a decent city. We consider different sizes of networks varying between 1000 and 100000 sensor nodes. The distribution of the nodes follows the Poisson process as often used in the literature. However, other models can also be used e.g., those based on the Thomas and Matérn process. Sensor readings are sampled from a correlated random field that captures the correlation properties of most real-world phenomena.

We build 2d-trees for each of the networks and evaluate the performance of the Database2dtreeM. From a 2d-tree the CalculatedSemivariogramM computes the semivariances of various distances h. The binning approach we consider has $\Delta = 50\,\text{m}$. As an example, any pair of nodes having a distance varying between 0 m and 100 m contributes in calculating the semivariance for distance $h = 50\,\text{m}$. We consider in total 17 distances varying between 50 and 7050 m. In order to closely follow the shape of the semivariogram before reaching its sill value, we generate the semivariances for consequent bins. After reaching the sill value we content to calculate the semivariances of few sparse bins. We realize the parallel processing by launching 17 parallel jobs each composed of a single mapper and a single reducer, and is in charge of calculating the semivariance of a bin. After finding the fitting model for the semivariogram, we then evaluate the performance of the SpatialpredictionM. We randomly discard temperature readings for different percentages of spatial location. The processing load for the prediction application is distributed on different jobs, each interpolating the temperature values of a set of 100 spatial location.

The envisaged scenario imposes some Hadoop configuration parameters. A single mapper processes the sensor data of the chosen region in its entirety in order to calculate the matrices Γ and γ. Thus, we fix the maximum size of a region to 250 MB enough for storing sensor data for 100000 nodes. The larger the number of nodes n in the network, the more memory is needed for storing the Γ matrix. This impacts the configuration of the heap-size for child jvms of maps/reduces which we fix to 600 MB adapted to the largest network size of 100000 nodes.

The Hadoop MapReduce programming model version 19.1 and HBase version 19.3 currently run on a cluster of 6 virtual

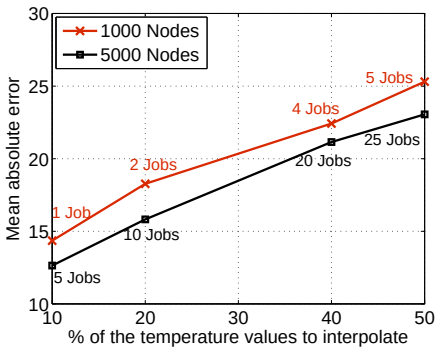

Figure 10: The mean absolute error function of the percentage of temperature readings to interpolate.

machines (VMs). The VMs are hosted by 3 workstations connected together by a 100 Mbps local-area network. Each workstation has a 2.66 GHz Quad-core processor. The operating system running of the machines is open SUSE 11.1, and each VM is allocated 3 GB of RAM.

5.2 Evaluating the modeling of a semivariogram

We recall that the purpose behind the Database2dtreeM is optimizing the search time in a given spatial network of nodes. In Figure 4 we show the required time for building a 2d-tree for different network sizes. As an example, forming a 2d-tree for a network of 50000 nodes requires only 659 s. These figures highlight two aspects: First, building a 2d-tree for significant sizes of networks in a sequential way is not a long process and thus, a parallel processing in this case is not of a high interest. Second, the building time of a 2d-tree is consumed only once. On the other hand, the output tree is used for each and every node in the network searching for predefined located neighbors. Thus, forming a tree and minimizing the number of checked nodes in the DB is meaningful. The CalculatedSemivariogramM generates the experimental semivariogram from a 2d-tree by launching 17 jobs running simultaneously on the cluster. Each job computes the semivariance of a single distance h. Figure 5 depicts the semivariogram of 5 different network sizes. We show uncertain semivariance values for the smallest network size of 1000 nodes, while for larger ones the curves converge to a sill of 31 having a range of 2500 m. The reason behind these phenomena can be seen in Figure 6. The figure shows the number of node pairs found at different distances h and participating in the calculation of the semivariance. The larger the number of nodes the more accurate the semivariance is. We take as an example the distance $h = 500$ m. The number of node pairs contributing in the calculation are 4, 2631, 15939, and 97017 respectively for the network sizes of 1000, 20000, 50000, and 100000 nodes. The low number of pairs in the smallest network explains the inaccuracy of its semivariance.

Figure 7 depicts the computation time of the cluster for generating a semivariogram and the time of the slowest job to finish successfully. The first is the summation of the time consumed by the 17 jobs. The second reflects the time needed for a single semivariogram to be generated. As an example, the computation time for a network of 50000 nodes is 2253 s while the slowest job needs 177 s to finish. It is important to clarify the fact that the load of generating a semivariogram is not equally distributed on all the jobs. The jobs in charge of large distances h are responsible of locating the nodes of a considerable geographical region in the 2d-tree. The number of located nodes is large and needs to go through the filtering process. This results into a longer processing time for computing the semivariance. Figure 8 refines the scope on the network size of 50000 nodes and depicts the processing time of the 17 jobs indexed by the distance h they are in charge of. As an example, the job computing the semivariance for $h = 7050$ m consumes 177 s, more than the double processing time of the job in charge of $h = 350$ m which runs for 77 s.

5.3 Evaluating the spatial prediction

The evaluation of the SpatialpredictionM focuses on two aspects: First, the computation time of the cluster needed to interpolate the temperature values of different percentages of spatial locations depicted in Figure 9. Second, the mean absolute error of the interpolated temperature readings function of the same percentage of spatial locations depicted in Figure 10. Both aspects are studied for two sizes of networks 1000 and 5000 nodes. We note that larger networks are possible as well, but we stress the fact that the Γ matrix increases in size resulting into slow mappers. The parallel processing in this module is realized by launching one job for each 100 temperature values to interpolate. It can be seen from both figures that the number of jobs varies with the size of the network as well as with the percentage of missing temperature values. As an example, interpolating 40% of the values in a network of 5000 nodes results into predicting 2000 values which requires 20 jobs.

In Figure 9 we reveal the hidden analysis behind the behavior of the computation time. When the percentage of missing temperature values increases, the number of jobs increases as well. On the other hand, the number of existing temperature values decreases leading to smaller Γ matrices which alleviates the computation time of the cluster. It can be seen from the figure that this phenomenon results into a global increase of the computation time until the percentage of missing temperature values reaches 50%. At this turnover point, the Γ matrices become much smaller in comparison to the previous cases leading into significant quicker mappers/jobs. The latter explains the sudden enhancement in the computation time of the cluster.

In Figure 10 we point out a larger mean absolute error for the network of 1000 nodes. This is due to the small number of nodes contributing in the interpolation. We take as an example the percentage of 20, the error in this case is 18.26 for the smallest network while it is 15.81 for the network of 5000 nodes. Both curves follow a similar behavior, the higher the percentage of values to interpolate, smaller is the number of nodes contributing in the interpolation resulting into a larger mean absolute error.

6. CONCLUSIONS AND DISCUSSIONS

In this paper we analyzed the processing of the sensor data after it has been received from the WSNs and stored on servers. We proposed a rich data model structuring the sensor information stored allowing a wide range of analyses. We provided a complete view on a MapReduce framework targeting the implementation of four categories of sensor data analyses: The acquisitional, aggregate, range, and spatio-temporal analyses.

As a start towards a complete framework, we implemented a spatio-temporal analysis mode as a specific example of spatial interpolation of phenomena to be observed with large-scale WSNs. We showed that Mapreduce can be used efficiently to handle large-scale spatio-temporal data that naturally arise from WSNs. The required workload is non-trivial and highly depends on the mathematical calculation of large matrices. Our experiments and performance measurements showed clearly that many operations must be done not locally to the WSN but in sever sites. In fact, for large-scale networks a considerable amount of computing power is required and there are clear limits what can be done in "real-time" fashion.

In our future work we aim at contributing an open source MapReduce library for sensor data analyses. Thus, we shall present as well the implementation and evaluation of the proposed framework package.

Acknowledgments

We acknowledge the partial financial support received from European Union (PSIRP-project) and from DFG (Deutsche Forschungsgemeinschaft) through UMIC Research Centre at RWTH Aachen University.

7. REFERENCES

[1] Apache Commons. `http://commons.apache.org/math/`, February 2010.

[2] Cambridge Mobile Urban Sensing (CamMobSens). `http://www.escience.cam.ac.uk/mobiledata/`, February 2010.

[3] Hadoop. `http://hadoop.apache.org/`, February 2010.

[4] HBase. `http://hadoop.apache.org/hbase/`, February 2010.

[5] Hypertable. `http://www.hypertable.org/`, February 2010.

[6] Oracle Real Application Clusters. `http://www.oracle.com/technology/products/database/clustering`, February 2010.

[7] A. Cary, Z. Sun, V. Hristidis *et al.* Experiences on Processing Spatial Data with MapReduce. *Scientific and Statistical Database Management*, 55699/2009:302–319, May 2009.

[8] C. Jardak, K. Rerkrai, A. Kovacevic *et al.* Design of Large-scale Agricultural Wireless Sensor Networks: Email from the Vineyard. *IJSNET*, 8, 2010.

[9] C. K. Baru, G. Fecteau, A. Goyal *et al.* . DB2 Parallel Edition. *IBM Systems Journal*, 34:292–322, 1995.

[10] F. Chang, J. Dean, S. Ghemawat *et al.* Bigtable: A Distributed Storage System for Structured Data. In *Proc. of the Symposium OSDI*, pages 205–218, Seattle, WA, November 2006.

[11] F. Oldewurtel, and P. Mähönen. Analysis of Enhanced Deployment Models for Sensor Networks. In *Proc. of VTC*, Taipei, Taiwan, 2010. accepted.

[12] F. Oldewurtel, J. Riihijärvi, and P. Mähönen. Impact of Correlation in Node Locations on the Performance of Distributed Compression. In *Proc. of WONS*, pages 135–142, Snowbird, USA, 2009.

[13] I. Chatzigiannakis, V. Liagkou, and P. Spirakis. A Trusted Architectural Model for Interconnecting Testbeds of Wireless Sensor Networks. In *Proc. of the Sympostiom ELMAR*, pages 515–518, Zadar, Croatia, September 2008.

[14] J. Ansari, J. Riihijärvi, P. Mähönen *et al.* Implementation and Performance Evaluation of nanoMAC: A Low-Power MAC Solution for High Density Wireless Sensor Networks. *IJSNET*, 2:341–349, 2007.

[15] J. Ansari, J. Sanchez, M. Petrova *et al.* Flexible Hardware/Software Platform for Tracking Applications. In *Proc. of VTC*, pages 6–10, Dublin, Ireland, April 2007.

[16] J. Dean, and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proc. of the Symposium OSDI*, pages 137–150, San Francisco, California, USA, December 2004.

[17] J. L. Bentley. Multidimensional Divide and Conquer. *Communications of the ACM*, 23:214–229, 1980.

[18] J. N. Al-karaki, and A. E. Kamal. Routing Techniques in Wireless Sensor Networks: a Survey. *IEEE Wireless Communications*, 11:6–28, March 2006.

[19] K. Aberer, M. Hauswirth, and A. Salehi. Infrastructure for Data Processing in Large-scale Interconnected Sensor Networks. In *International conference on mobile data management*, pages 198–205, Mannheim, Germany, May 2007.

[20] L. G. Valiant. A Bridging Model for Parallel Computation. *Communications of the ACM*, 33:103–111, August 1990.

[21] M. K. Vairamuthu, S. Nesamony, M. E. Orlowska *et al.* Investigative Queries in Sensor Networks. *Lecture notes in computer science*, 4537/2010:111–121, August 2007.

[22] N. A. C. Cressie. *Statistics for Spatial Data*. John Wiley and Sons, 1991.

[23] P. Aoki, R.J. Honick, A. Mainwaring *et al.* A Vehicle for Research: Using Street Sweepers to Explore the Landscape of Environmental Community Action. In *Proc. of the International Conference on Human Factors in Computing Systems*, pages 375–384, Boston, MA, USA, April 2009.

[24] R. Bose. Sensor Networks-Motes, Smart Spaces, and Beyond. *IEEE Pervasive Computing*, 8:84–90, September 2009.

[25] R. H. Arpaci-Dusseau, E. Anderson, N. Treuhaft *et al.* Cluster I/O with River: Making the Fast Case Common. In *Proc. of the Workshop IOPADS*, pages 10–22, Atlanta, Georgia, May 1999.

[26] W. Gropp, E. Lusk, and A. skjellum. Using MPI: Portable Parallel Programming with the Message-Passing Interface. In *MIT Press*, Cambridge, MA, 1999.