An Architecture For High-performance Privacy-preserving And Distributed Data Mining

2009

James Secretan
University of Central Florida

Find similar works at: https://stars.library.ucf.edu/etd

University of Central Florida Libraries http://library.ucf.edu

Part of the Computer Engineering Commons

STARS Citation

Secretan, James, "An Architecture For High-performance Privacy-preserving And Distributed Data Mining" (2009). Electronic Theses and Dissertations. 3987.
https://stars.library.ucf.edu/etd/3987

This Doctoral Dissertation (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of STARS. For more information, please contact lee.dotson@ucf.edu.
An Architecture for High-Performance Privacy-Preserving and Distributed Data Mining

by

Jimmy Secretan
B.S. University of Central Florida, 2004
M.S. University of Central Florida, 2007

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the School of Electrical Engineering and Computer Science in the College of Engineering and Computer Science at the University of Central Florida Orlando, Florida

Fall Term 2009

Major Professor:
Michael Georgiopoulos
Abstract

This dissertation discusses the development of an architecture and associated techniques to support Privacy Preserving and Distributed Data Mining. The field of Distributed Data Mining (DDM) attempts to solve the challenges inherent in coordinating data mining tasks with databases that are geographically distributed, through the application of parallel algorithms and grid computing concepts. The closely related field of Privacy Preserving Data Mining (PPDM) adds the dimension of privacy to the problem, trying to find ways that organizations can collaborate to mine their databases collectively, while at the same time preserving the privacy of their records. Developing data mining algorithms for DDM and PPDM environments can be difficult and there is little software to support it. In addition, because these tasks can be computationally demanding, taking hours of even days to complete data mining tasks, organizations should be able to take advantage of high-performance and parallel computing to accelerate these tasks. Unfortunately there is no such framework that is able to provide all of these services easily for a developer. In this dissertation such a framework is developed to support the creation and execution of DDM and PPDM applications, called APHID (Architecture for Private, High-performance Integrated Data mining). The architecture allows users to flexibly and seamlessly integrate cluster and grid resources into their DDM and PPDM applications. The architecture is scalable, and is split into highly de-coupled
services to ensure flexibility and extensibility. This dissertation first develops a comprehensive example algorithm, a privacy-preserving Probabilistic Neural Network (PNN), which serves a basis for analysis of the difficulties of DDM/PPDM development. The privacy-preserving PNN is the first such PNN in the literature, and provides not only a practical algorithm ready for use in privacy-preserving applications, but also a template for other data intensive algorithms, and a starting point for analyzing APHID’s architectural needs. After analyzing the difficulties in the PNN algorithm’s development, as well as the shortcomings of researched systems, this dissertation presents the first concrete programming model joining high performance computing resources with a privacy preserving data mining process. Unlike many of the existing PPDM development models, the platform of services is language independent, allowing layers and algorithms to be implemented in popular languages (Java, C++, Python, etc.). An implementation of a PPDM algorithm is developed in Java utilizing the new framework. Performance results are presented, showing that APHID can enable highly simplified PPDM development while speeding up resource intensive parts of the algorithm.
To my wife, who endured as much as I did for this degree.
ACKNOWLEDGMENTS

This work was supported in part by a National Science Foundation Graduate Research Fellowship as well as National Science Foundation grants: 0341601, 0647018, 0717674, 0717680, 0647120, 0525429, 0203446.

I would like to thank my committee members, and my advisor Dr. Michael Georgiopoulos. In them, I have found not only the aid of mentors but the advice of friends.

I would also like to thank my fellow grad students in the Machine Learning Lab. With their support, I was never on my journey alone.

To my parents, who inspired me with a love of learning, and whose pride provided motivation enough when I got discouraged.

To my wife, who has sacrificed her time with me, because she believed that I could make our lives and the lives of others better. Angela, I love you more than you’ll ever know.

To the Maker, whose profoundly complex creation constantly inspires us.
## TABLE OF CONTENTS

LIST OF FIGURES ........................................... xii

LIST OF TABLES ............................................. xvi

CHAPTER 1: INTRODUCTION ................................. 1

CHAPTER 2: PRIVACY PRESERVING DATA MINING .......... 4

  2.1 Data Perturbation Approaches .......................... 5
  2.2 Signal Processing Approaches .......................... 6
  2.3 Distributed Meta-Learning Approaches ................. 6
  2.4 Secure Multi-party Computation Techniques (SMC) ..... 8
  2.5 Current SMC-based Privacy Preserving Data Mining Algorithms .... 10

CHAPTER 3: COMMON SMC TECHNIQUES ................. 12

  3.1 Secure Sum ............................................ 12
  3.2 Secure Scalar Product .................................. 15
  3.3 Secure Tree Operations .................................. 17

CHAPTER 4: DESIGNING SMC-PPDM ALGORITHMS ....... 20
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.0.1</td>
<td>Partitioning</td>
<td>22</td>
</tr>
<tr>
<td>4.0.2</td>
<td>Adversarial Models</td>
<td>23</td>
</tr>
<tr>
<td>4.0.3</td>
<td>Privacy / Performance Tradeoffs</td>
<td>24</td>
</tr>
<tr>
<td>5.1</td>
<td>The PNN Algorithm</td>
<td>30</td>
</tr>
<tr>
<td>5.2</td>
<td>Privacy Preserving Distributed PNN Algorithm</td>
<td>32</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Distributed Model and Assumptions</td>
<td>34</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Determining $\sigma$</td>
<td>35</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Horizontally Partitioned Databases, Public Queries</td>
<td>36</td>
</tr>
<tr>
<td>5.2.4</td>
<td>Horizontally Partitioned Databases, Private Queries</td>
<td>40</td>
</tr>
<tr>
<td>5.2.5</td>
<td>Vertically Partitioned Databases, Public Queries</td>
<td>48</td>
</tr>
<tr>
<td>5.2.6</td>
<td>Vertically Partitioned Databases, Private Queries</td>
<td>55</td>
</tr>
<tr>
<td>5.2.7</td>
<td>Algorithm Summary</td>
<td>60</td>
</tr>
<tr>
<td>5.3</td>
<td>Experimental Setup</td>
<td>61</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Implementation</td>
<td>61</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Experimental Database</td>
<td>62</td>
</tr>
<tr>
<td>5.3.3</td>
<td>Test Description</td>
<td>62</td>
</tr>
<tr>
<td>5.4</td>
<td>Results</td>
<td>63</td>
</tr>
<tr>
<td>5.5</td>
<td>Summary</td>
<td>67</td>
</tr>
</tbody>
</table>
CHAPTER 9: INEXPENSIVE AND EFFICIENT ARCHIVAL STORAGE FOR GRID AND DISTRIBUTED DATA MINING

9.1 Redundant and High Performance Distributed File Systems

9.2 Grid Storage and Computation

9.3 Grid Economics

9.4 An Algebra of Storage Resource Composition

9.5 Resource allocation algorithms

9.5.1 Redundant Composition Algorithm

9.5.2 Additive Composition Algorithm

9.5.3 Distributed Error Composition Algorithm

9.6 Experimental study

9.6.1 Experimental setup

9.6.2 Results

CHAPTER 10: CONCLUSION

CHAPTER 11: FUTURE WORK

11.1 Enhancements to APHID
# LIST OF FIGURES

3.1 Diagram of the secure sum for \( K \) parties. .......................... 14
3.2 SPMD-style code for the simple secure sum. ............................ 14
3.3 Secure Scalar Product algorithm. ........................................... 17
3.4 Tree structured secure operation. .......................................... 18
3.5 Tree-based algorithm for secure associative operations .............. 19
4.1 A flowchart of a possible SMC-PPDM design process. ............... 21
4.2 Vertical, horizontal and arbitrary partitioning. ....................... 22
5.1 General architecture of the Probabilistic Neural Network .............. 33
5.2 Pseudo-code for the PNN algorithm (serial version) ............... 34
5.3 The PP-PNN algorithm for the horizontally partitioned public query case 37
5.4 The PP-PNN Algorithm for the horizontally partitioned private query case 44
5.5 The PP-PNN Algorithm for the vertically partitioned public query case . 49
5.6 The PP-PNN Algorithm for the vertically partitioned private query case . 56
5.7 Performance of the horizontally partitioned, public query PNN ........ 64
5.8 Performance of the horizontally partitioned, private query PNN ........ 65
9.9 Percentage of compositions for runs at different average request sizes. . . 138

B.1 Example of the secure sum for 3 parties. .............................. 151
B.2 $P^1$ sends the encrypted operands to $P^2$. .............................. 152
B.3 $P^2$ performs encrypted operations and sends back the encrypted scalar
  product to $P^1$ .................................................. 153
B.4 Leaf parties encrypt their values and send them to their parents. . . . . 154
B.5 $P^2$ and $P^3$ add their two operands to the ones received. ............... 155
B.6 The sum is shared between parties $P^1$ and $P^4$ .......................... 155
B.7 Setup of the horizontally partitioned PNN problem ....................... 156
B.8 Adding the CCP values for $c_1$ ............................................. 157
B.9 Calculating the SSP between the test point and a training point. ....... 158
B.10 Calculating the remainder of the shares between parties. ................. 159
B.11 The final SSP to yield the CCPs. ......................................... 159
B.12 The setup for the vertically partitioned examples. ....................... 160
B.13 Calculating distance shares for the vertically partitioned public query
  PNN ...................................................................... 161
B.14 The shared secure sum of distance values in the vertically partitioned,
  public query PNN ................................................... 162
B.15 Padding and performing the SSP with the CCPs in the vertically parti-
  tioned public query PNN ............................................. 162
B.16 Calculating the encrypted shares for the vertically partitioned, private query PNN. ................................. 163

B.17 Encrypted shared secure sum for the vertically partitioned, private query PNN. .............................................. 164
**LIST OF TABLES**

5.1 Comparison of SMC-PPDM Classification Algorithms .................................. 30
5.2 Comparison of PP-PNN Algorithms ............................................................. 61
5.3 Average Time Integer and NTL Integer Arithmetic ......................................... 66
8.1 Types of variable sharing policies. ............................................................... 90
8.2 Examples of supported operations at the PPDM level. ..................................... 94
9.1 Notations for Analysis ..................................................................................... 114
A.1 Notations for Analysis, chapter 3 ............................................................... 147
A.2 Notations for Analysis, Section 5.1 ............................................................. 147
A.3 Notations for Analysis, Section 5.2.1 ........................................................... 148
A.4 Notations for Analysis, Sections 5.2.3 and 5.2.4 ........................................... 148
A.5 Notations for Analysis, Sections 5.2.5 and 5.2.6 ........................................... 148
A.6 Pseudocode Functions ..................................................................................... 149
CHAPTER 1
INTRODUCTION

Modern organizations manage an unprecedented amount of data, which can be mined to generate valuable knowledge, using several available data mining techniques. While data mining is useful within an organization, it can yield further benefits with the combined data of multiple organizations. And, in fact, many organizations are interested in collaboratively mining their data. This sharing of data, however, creates many potential privacy problems. Many organizations, such as health organizations, have restrictions on data sharing. Businesses may be apprehensive to share trade secrets despite the value of cooperative data mining. At the same time, privacy concerns for individuals are rapidly gaining attention. Instead of doing away entirely with the prospect of cooperative data mining, research has instead focused on Privacy Preserving Data Mining (PPDM), which uses various techniques, statistical, cryptographic and others, to facilitate cooperative data mining while protecting the privacy of the organizations or individuals involved.

However, PPDM research is still in its infancy, and there is a lack of practical systems currently in use. Even if organizations currently have the legal infrastructure in place for sharing data, there is a lack of developmental support for PPDM systems. Organizations currently trying to implement PPDM systems would face a lack of available toolkits, libraries, middleware and architectures that are ready for deployment. The costs involved are potentially high, because of the lack of familiarity with PPDM technology. In ad-
dition, because complex computation is often required, high performance and parallel computing technologies are necessary for efficient operation, adding yet another level of complexity to development. The purpose of this research is to provide an architecture and development environment that will allow organizations to easily develop and execute PPDM software. By borrowing from familiar parallel paradigms, the architecture aims to ease the introduction of PPDM technology into the existing database infrastructure. Furthermore, the system intends to seamlessly integrate high performance computing technologies, to ensure an efficient data mining process.

Because of extensive communication over relatively slow wide area networks, and because of the large computational requirements of cryptographic and other privacy-oriented technologies, resource requirements for PPDM algorithms can be intense. One study [VC04b] describes building a 408-node decision tree from a 1728 item training set in 29 hours. While there is much research that discusses available algorithms and techniques in PPDM, few studies focus on high-performance computational architectures that support them. Therefore, this research presents a development environment and runtime system specifically geared toward PPDM.

The contributions of this dissertation are: (1) an analysis of the shortcomings of current software to support PPDM algorithm development, (2) middleware for managing the execution of PPDM algorithms across multiple organizations, (3) the integration of high performance and parallel computing middleware into the PPDM execution environment, (4) a framework for easily developing PPDM software, (5) a new suite of privacy-preserving data mining algorithms for the Probabilistic Neural Network (PNN),
and (6) a set of new data scheduling algorithms to support more efficient storage and computation for grid-based high-performance data mining.

The layout of this dissertation is as follows. Chapter 2 gives a broad overview of the field of privacy preserving data mining, and chapter 3 describes some techniques commonly used to support PPDM algorithms. Chapter 4 describes the general process of designing a PPDM algorithm that relies on secure multiparty computation (SMC) techniques. This design knowledge is then put into practice in chapter 5 where a suite of privacy-preserving algorithms for the data intensive PNN is presented. The exploration of the suite of PNN algorithms gives us a perspective on the shortcomings of current frameworks in supporting PPDM algorithm development, and these lessons learned are discussed in chapter 6. Chapter 7 reviews work done in high-performance and distributed data mining. The work lays the foundation necessary to understand how high-performance computing resources can be brought to bear in supporting PPDM algorithms. In chapter 8 system called APHID (Architecture for Private and High-performance Integrated Data mining) is described, which addresses the limitations described in chapter 6. In chapter 9, the idea of low-cost composition of grid storage resources to support archival storage of databases and mining output is discussed. These techniques have the potential to further decrease costs for data mining operations. Finally, in chapter 10, we summarize the work, and suggest future directions in chapter 11. Appendix A summarizes the notation used in the dissertation, and appendix B offers easy to follow pedagogical examples for some of the SMC techniques employed in the dissertation.
CHAPTER 2
PRIVACY PRESERVING DATA MINING

Literature in the field of Privacy Preserving Data Mining (PPDM) concentrates on how to coherently combine and mine databases so as to preserve the privacy of the individual parties’ data. How this is accomplished, and the extent to which it is accomplished, differs in the various branches of PPDM. First, there are data perturbation methods, which focus on obfuscating the original data using random perturbations, but doing it so that the original distributions of the data can be easily recovered. Another, similar area involves using signal processing techniques to create approximations of the data distributions, which more effectively preserve privacy than access to the raw data. A very different approach uses Secure Multi-party Computation (SMC) to compute the data mining functions, often in a more exact, more private way (at the expense of efficiency). There are a few approaches that utilize what we will refer to as Distributed Meta-Learning. These mostly involve training private classifiers on private data and then using voting or some other means to combine the classifier or estimator outputs. Finally, there are a few approaches which use some combination of methods from these 4 major approaches. Although, the areas of anonymization and Private Information Retrieval (PIR) are related, we will not include them here to simply focus on more germane systems.
2.1 Data Perturbation Approaches

The data perturbation approach for data mining, largely pioneered in [AS00], involves perturbing the values of the training data records in a way such that the individual records are not identifiable while at the same time using methods to recover the original distribution. In this work, the authors discussed a privacy-preserving decision tree, which, depending on the level of privacy, could produce a decision tree with accuracy close to the decision tree trained on the unperturbed data. The privacy metric used in the paper measures the confidence interval for the perturbed values. Here, two different methods are used for perturbation. The first is *Value Distortion*. To distort the values, a random value $r$ is added to the value of the dimension $x_i$. The second is referred to as *Value-Class Membership*. This is where values are partitioned into more general classes. For instance, imagine a survey where the user is asked for his age. Privacy would be better preserved by asking the user to check a range (e.g. 30-50 years old) instead of supplying the exact numerical age. In [ESA02], the authors reconstruct randomized records to produce association rules. Approaches from [AS00] and [ESA02] were later generalized into a framework called FRAPP (FFramework for Accuracy in Privacy-Preserving mining) [AH05]. FRAPP casts the prior approaches to designing perturbation methods into matrix-theoretic terms.
2.2 Signal Processing Approaches

The approach in [MCG06] involves transforming the data using a secure Fourier type operation and then providing an incomplete set of coefficients, to increase both privacy and performance. The authors present algorithms for both horizontally and vertically partitioned cases, which generate an approximation of the distributed data, the models of which can be fed into any data mining algorithm. The transforms that the algorithm performs also have the benefit of largely preserving Euclidean distances between points, to allow more accurate reconstruction of the data. Efficiency and privacy are potentially traded off with accuracy; fewer coefficients of the transform represent more privacy and efficiency, at the potential cost of accuracy. The signal processing approaches suffer from some of the same difficulties in the data perturbation methods. They require that accuracy to be traded for privacy.

2.3 Distributed Meta-Learning Approaches

The area of distributed meta-learning PPDM has already produced useful, production quality systems. While distributed meta-learning approaches are not explicitly privacy preserving, many have the very useful side-effect of efficiently and almost perfectly preserving privacy. In [PC00], the authors use a voting method based on classifiers that are constructed on individual collections of the data. This is system is based on the Java Agents for Metalearning system. This is a sophisticated system which can import classifiers from other hosts, and utilize what are called bridging methods for bringing
together databases that are not immediately compatible. In [TBV04], the authors utilize a web services based environment that combines the output of distributed classifiers using certain expert system rules. These methods fall under the umbrella of what is called meta-learning. In the single party sense, meta-learning methods have usually involved training classifiers with different parameters or different algorithms on a single database. In the distributed data mining sense, it has begun to encompass the incorporation of distributed classifiers with potential access to many different databases.

These distributed meta-learning methods have many distinct advantages over other PPDM approaches. To begin with, they are typically computationally efficient, even compared to non-privacy preserving DDM algorithms. The communication requirements are very small; usually, all that must be communicated is the final classification of each party, making communication on the order of the number of parties. This is more efficient than many SMC methods and even many random perturbation methods. In addition, the privacy is easy to maintain and understand, unlike many SMC methods. Finally, the methods presented in [PC00] and [TBV04] are technically very mature. These systems use agent technologies and web services, which are quickly gaining notoriety as the new de facto standards for producing distributed data mining software.

The disadvantage to distributed meta-learning is that, because classifiers are not built globally on data, the model’s performance may suffer as a result of incomplete information.
Secure Multi-party Computation Techniques (SMC)

Secure Multiparty Computation (SMC) involves the use of cryptographic techniques to ensure almost optimal privacy. SMC as a field grew out of work to solve the Millionaire’s Problem [Yao86]. The Millionaire’s Problem is as follows: two millionaires wish to find out who has more money, but neither wants to disclose his/her individual amount. Yao proved that there was a secure way to solve this problem by representing it as a circuit which shares random portions of the outputs. Later it was proved in [GMW87] that any function could be securely computed using this kind of arrangement. However, using Yao circuits is typically inefficient. The problem must be represented as a circuit, which may be large, especially for complex data mining algorithms. The circuit also must have inputs for all of the inputs of the secure algorithm, making the circuit potentially enormous for large numbers of inputs. This is typically the case in large scale data mining. In distributed environments, the computation and communication requirements can make this kind of paradigm almost impossible to practically use for all but the smallest sub problems. This gives rise to more specific solutions using more efficient cryptographic techniques.

One of the earliest and most basic cryptographic operations used in PPDM is the secure sum [Sch95]. The secure sum is a simple technique for allowing multiple parties to add together a collection of numbers by having one of the parties add and then later remove a random number.

Another basic primitive of PPDM is oblivious transfer [Rab81, EGL85, NP99]. The most general instance of oblivious transfer, 1-out-of-N oblivious transfer, allows one party
to select only one piece of available data from a second party without allowing the second party to know which piece of data the first selected. For instance, in Oblivious Polynomial Evaluation, $S$ has a polynomial $P$ of degree $k$ over a field $F$. With Oblivious Polynomial Evaluation, $R$ can learn the value of $P(x)$ where $x \in F$ without having $S$ learn the value of $x$. In [Pin02], the authors posit oblivious transfer as a primary building block for SMC, and present an ID3 algorithm that utilizes oblivious polynomial evaluation. [HH06] presents a secure algorithm for computing distances to use in secure kernel methods, using oblivious transfer. While our work does not make use of oblivious transfer, a discussion of SMC is incomplete without it.

A technique which provides a building block for many SMC algorithms is the set of additively homomorphic encryption algorithms. By definition, an encryption algorithm is additively homomorphic if the following holds true:

$$Enc(A) \otimes Enc(B) = Enc(A + B)$$

where $Enc$ is the encryption function of the cryptosystem which also defines an operation $\otimes$. Given this operation, it is possible for two parties to participate in computation together without compromising their operands. The two most popular and seminal systems for homomorphic encryption were provided by [Pai99] and [DJ01].

Cryptographic techniques have also been integrated into larger building blocks in order to implement complex data mining algorithms. One of the most basic of these primitive operations is the scalar product. Privacy preserving scalar product protocols are suggested in [DA01], [VC02] and [IGA02]. However, later cryptanalysis in [GLL04]
reveals that the protocols in [DA01] and [VC02] were insecure in certain cases, and
presents an improved algorithm for finding the scalar product. This new scalar product
algorithm has been used, or at least been suggested for use, in several different SMC-
PPDM algorithms including [JPW06], [YW05], [JW05] and [YJV06a].

The authors in [CKV03] propose a toolkit of components for addressing privacy preser-
vation. In their paper, they survey techniques for privacy preserving decision trees, k-NN,
ARM, and Naïve Bayes. The proposed toolkit provides the following set of privacy primi-
tives, a secure sum, a secure set union, a secure size of set intersection and a secure scalar
product, arguing that many PPDM algorithms can be developed using those techniques.

2.5 Current SMC-based Privacy Preserving Data Mining Algorithms

SMC-PPDM methods often involve the redesign of existing algorithms using SMC tech-
niques. Again, this is because Yao circuits, while general as as solution, require compu-
tation and communication on the order of the number of inputs and complexity of the
algorithm. In a number of data mining situations, the inputs are typically many and the
algorithms are quite complex. Therefore, it is advantageous to seek more efficient solu-
tions of implementing them in a privacy preserving distributed setting. When designing
an SMC-PPDM algorithm, it is common to utilize several of the techniques discussed in
the previous section. By properly combining these techniques, one can produce a privacy
preserving algorithm with greater efficiency.
Many machine learning algorithms have been recast as into privacy preserving versions. These include decision trees (vertically partitioned in [DZ02a] and horizontally partitioned in [Pin02] and [LP02]), the Naïve Bayes classifier (vertical [VC04a] and horizontal [KV03]), Bayesian networks (vertical [WY04], [YW05]), General Clustering (horizontal [ISS06]), k-Means Clustering (vertical [VC03a] and arbitrary [JW05]), Support Vector Machines (vertical in [YV] [YVJ06a] [YVJ06b], horizontal in [YJV06b]), k-Nearest Neighbor (horizontal in [KC04b] [XCL06], vertical in [ZCM05]), Association Rule Mining (vertical in [VC02] [HLH05], horizontal in [KC04a]), linear regression (vertical in [DA01] [DHC04] [KLS05], horizontal in [KLS05]), EM (vertical in [RKK04]). In [BOP06], the authors present some techniques for computing a privacy preserving neural network. However, in [BOP06], the authors only discuss maintaining the privacy of the query for one party, and the privacy of the network for another party. They do not involve training with data from multiple parties. The authors of [JPW06], present a new clustering algorithm, designed from the ground up with privacy preservation in mind.

Then there are some hybrid methods that combine techniques from SMC and perturbation approaches. One such example is the privacy preserving k-NN for vertically partitioned data sets presented in [ZCM05].
CHAPTER 3
COMMON SMC TECHNIQUES

In this chapter, some functions that are common to many of the SMC-PPDM algorithms are described. These include the secure sum, the secure scalar product and tree-based secure operations.

3.1 Secure Sum

As mentioned before, a fundamental algorithm in SMC is the secure sum [Sch95]. The secure sum works as follows. Suppose that parties \( P^1 \) to \( P^K \) each have a value \( v^k \), and they wish to compute the sum \( v = \sum_{k=1}^{K} v^k \). Furthermore, the sum \( v \) is limited to the range \([0...F]\). Without loss of generality, \( P^1 \) starts by choosing a random number \( R \) from within the range \([0...F]\). \( P^1 \) then calculates \((v^1 + R) \mod F\) and sends this to \( P^2 \). The modulus by \( F \) is necessary to ensure that the sum is kept randomly distributed within the range \([0...F]\), therefore yielding no additional information to any observer. Parties \( P^2 \) through \( P^K \) repeat the calculations of \( P^1 \). Finally, \( P^K \) sends its sum back to \( P^1 \). \( P^1 \) then adds \(-R\) to the sum and takes the \( mod F \) of the sum, yielding \( v = \sum_{k=1}^{K} v^k \). To see why this works, we consider two cases. In the first case, the total plus the random number, \( v + R < F \). In this case, the modulus never changes the sum, and it is clear how subtracting \( R \) will yield \( v \). In the second case, \( v + R \geq F \). Because of the way that \( R \) is
chosen, it is known that $v + R \leq 2F$. Therefore, the modulus will affect the sum $v + R$ only once, making it $v + R - F$. At the end, $R$ is subtracted making this sum $v - F \leq 0$. Applying a modulus $F$ at this point will yield $v$, the desired sum. In [YV04], the authors extend this to a secure matrix sum. The extension is simple, with $R$, becoming a matrix of random numbers drawn from the range $[0...F]$. For a concrete example of the secure sum, see the appendix, section B.1. The secure sum becomes especially useful in adding probabilities across parties in the PNN algorithms. However, to use real numbers instead of integers requires mapping them to a fixed-point notation.

A pictorial illustration of how the secure sum is computed is shown in figure 3.1. Pseudocode in Single Program Multiple Data (SPMD) style is given in figure 3.2. This style is used in many parallel programming toolkits, including MPI. In SPMD style, a single program is run on all parties, with different branches based on the party’s individual value of $k$. In the pseudo-code, lines 1–4 are executed at $P^1$, where they add a random value (line 1), send this value to the next party (line 2) and finally, receive the sum from $P^K$ and subtract the original random value. At every other party, lines 5–6 are executed, receiving the partial sum from the previous party, and sending the previous sum plus its own value to the next.

The PNN algorithm in chapter 5 utilizes an extension to the secure sum which employs secret sharing ([Sha79] and [Bla79]), called a shared secure sum. In this case, it is desirable to split the final result of the secure sum between two parties in such a way that adding their new shares yields the final sum, but neither party has any significant knowledge of the full sum.
Figure 3.1: Diagram of the secure sum for $K$ parties.

| Input: | Parties $P_1$ through $P_K$, each with values $v_1$ through $v_K$ respectively |
| Output: | The sum $v = \sum_{k=1}^{K} v_k$ on party $P_1$ |
| 1 if $k == 1$ then |
| 2 | $R = \text{rand}() \mod F$; |
| 3 | $\text{send}(2, (v_1 + R) \mod F)$; |
| 4 | $v = \text{receive}(K) - R \mod F$; |
| else |
| 5 | $\text{send}(k \mod (K + 1), \text{receive}(k - 1) + v_k \mod n)$; |

Figure 3.2: SPMD-style code for the simple secure sum.

One such way to implement this is as follows. Suppose $P^A$ and $P^B$ wish to share the sum of values from all parties, $v = \sum_{k=1}^{K} v_k$. As in the standard secure sum, each party $P^k$ has a share $v^k$ of the sum. All of the parties begin by splitting their respective shares into two shares $v^k_1$ and $v^k_2$. This splitting of shares is done by drawing a random number $R$ from the range $[0...F]$ then calculating the two new shares as $v^k_1 = R$ and $v^k_2 = v^k - R$.

At each party, we then add $F$ to both $v^k_1$ and $v^k_2$ to ensure that each share is greater than or equal to 0, as is required by the secure sum.

After this, two separate secure sums are conducted, one beginning and ending at $P^A$ and one beginning and ending at $P^B$. These secure sums are conducted over the range $[0...(K + 1)F]$, because $F$ has been added to every one of $K$ shares to ensure that those
shares remain positive. At the end of each secure sum, the additional $KF$ that was the result of adding $F$ to each share is subtracted, yielding the appropriate share of the sum to each respective party.

The shared secure sum is also secure, because by the composition theorem [Gol98], it is composed of two secure sums, which have already been shown secure. The quantity $KF$ is public knowledge and therefore its addition does not breach security. This method is only one of the many possible ways to effect a shared secure sum.

### 3.2 Secure Scalar Product

In this work, we will have frequent need of a secure scalar product, and therefore, we use the one from [GLL04]. This protocol is built around an additively homomorphic cryptosystem. By definition, an encryption algorithm is additively homomorphic if the following holds true: $\text{Enc}(A) \otimes \text{Enc}(B) = \text{Enc}(A + B)$ where $\text{Enc}$ is the encryption function and $\otimes$ is the homomorphic operation of the cryptosystem.

For our application, we use the cryptosystem provided by Paillier [Pai99]. It is a public key cryptosystem. As with other public key cryptosystems, a party can encrypt information using the public key $pk$, but a party needs the private key $sk$ in order to decrypt information. This way, one party can encrypt its data, and then have another party process the data without being able to decrypt it.

The Paillier cryptosystem’s homomorphic operation is implemented in terms of a modular multiplication. From this relation, an additional property is derived in [Pai99].
If we multiply the ciphertext by itself a certain number of times, it is equivalent to adding it the same number of times. Therefore, putting the ciphertext to the power of another plaintext will give the encrypted product, $Enc(A)^B = Enc(AB)$. Given this, it is possible for two parties to participate in computation together without compromising their operands.

In this work, we used the algorithm presented in [GLL04] which we repeat here. Two parties, $P^A$ and $P^B$, have the vectors $x^A$ and $x^B$ respectively. The secure scalar product (SSP) algorithm of these two vectors is depicted in figure 3.3. The algorithm works as follows. Party $P^A$ generates the public key ($pk$) and private key ($sk$), as Paillier’s cryptosystem is a public key cryptosystem (line 2). The public key is then sent to $P^B$ (line 3). $P^A$ then proceeds to encrypt all of the elements of its vector (lines 4-5), finally sending this vector of encrypted values to $P^B$ (line 6). $P^B$, receives this vector on line 9. On line 10, it generates a random number, $s^B$, to be designated as his share of the vector. On line 11, it takes each encrypted element, and sets it to the power of its plaintext elements. As stated before, this is equivalent to multiplying the numbers together. It then takes the product of all of these, thereby adding the products of the corresponding vector elements. Finally, it subtracts a random share from this product, giving $P^B$ its final random share of $s^B$. It then sends the remainder back to $P^A$ (received in line 7), where only $P^A$ can decrypt it, because it has the private key.

A detailed example of the secure scalar product can be found in appendix section B.2.
Input: Vectors $x^A$ and $x^B$ on parties $P^A$ and $P^B$ respectively.
Output: Random shares $s^A$ and $s^B$ on parties $P^A$ and $P^B$ respectively.

1. if $k == A$ then
2.   $(sk, pk) = \text{GenerateKeyPair}()$;
3.   send($P^B, pk$);
4.   foreach $i \in (1, \ldots, D)$ do
5.     $cx(i) = \text{Enc}_{pk}(x^A(i))$;
6.     send($P^B, cx$);
7.   $s^A = \text{Dec}_{sk}(\text{receive}(P^B))$;
8. if $k == B$ then
9.     $cx = \text{receive}(P^A)$;
10.    $s^B = \text{rand}()$;
11.    send($P^A, \prod_{i=1}^{D} cx(i)^{x^B(i)} \cdot \text{Enc}_{pk}(-s^B)$);

Figure 3.3: Secure Scalar Product (SSP) algorithm from [GLL04].

3.3 Secure Tree Operations

Both the secure sum and the homomorphic additions and multiplications can be expensive operations, because of the computational complexity of cryptographic operations and the relatively high latency and low bandwidth of Internet connections. In order to make certain secure functions faster, we utilize a concept presented in [VC03b]. The authors show that any function that can be represented as $y = f(x_1, \ldots, x_k) = x_1 \otimes x_2 \otimes \ldots \otimes x_k$ with $\otimes$ being an associative operation can be securely computed using a more efficient tree structure. Luckily, many operations fit this description (i.e. set union, intersection, multiplication, addition). In figure 3.4, we reiterate the structure of the communication as presented in [VC03b], with the code shown in figure 3.5. In figures 3.4 and 3.5, it is assumed that the parties are numbered from $P^1 \ldots P^K$ with their respective operands $v^1 \ldots v^K$ without loss of generality. The leaf parties ($P^{2^w}$ through $P^K$) begin by performing the secure operation $\otimes$ with their parents ($P^{2^{w-1}}$ through $P^{2^w-1}$). If, for instance, the
The operation to be performed is a tree-based secure sum, the leaf parties and their parents will participate in a three-way secure sum, as shown in figure 3.1. The result of this associative operation should end up on those parent parties and then the parent parties themselves participate in another three-way operation. These three-way operations continue up the tree until the results reach parties $P_1$, $P_2$ and $P_3$. For a detailed example of the tree operation used in the private query, vertically partitioned PNN, see appendix B.3.

![Figure 3.4: Tree structured secure operation.](image-url)

A detailed example of a secure tree-based operation can be found in appendix section B.3.
**Input:** An associative operation $\otimes$ to perform. At each party $P^k$, a value $v^k$. A party $P^1$ where the final result should end up.

**Output:** The final function value $y = f(x_1, \ldots, x_k) = x_1 \otimes x_2 \otimes \ldots \otimes x_k$ at $P^1$.

if $k == 1$ then
  1. $y = v^1 \otimes \text{receive}(P^2) \otimes \text{receive}(P^3)$;
else if $k \geq 2^\lceil \log_2 K \rceil$ then
  4. send($P^{[k/2]}, v^k$);
else
  6. send($P^{[k/2]}, \text{receive}(P^{2k}) \otimes \text{receive}(P^{2k+1}) \otimes v^k$);

Figure 3.5: A tree-based algorithm for secure associative operations modified from [VC03b]
CHAPTER 4
DESIGNING SMC-PPDM ALGORITHMS

Through examining the large body of SMC-PPDM algorithms development, we now present a high-level design process for designing SMC-PPDM algorithms. Figure 4.1 represents a typical SMC-PPDM development process. To begin with, the type of partitioning, that is, how the data is laid out, must be determined from the intended application. The next step is to determine if a similar algorithm exists in the literature. As is seen in section 2.5, there are many well known algorithms for which one or more SMC-PPDM versions have been established. Even if they are not immediately applicable, some can be modified or their techniques reused to suit the necessary algorithm. Existing algorithms for non-private distributed parties, grids and clusters may also provide design inspiration. In certain cases, a traditionally distributed version of an algorithm can be converted to a privacy preserving form with relatively little additional effort. One example comes the privacy preserving Probabilistic Neural Network [SGC07]. One version of this algorithm simply involved introducing a simple secure sum [Sch95] to the cluster computer version of the algorithm [SGM06]. Next in the design process, the designer must determine which SMC techniques and algorithms (like those in chapter 3) should be employed in the SMC-PPDM algorithm to be designed. The choice of SMC techniques will ultimately depend on the security required for the PPDM algorithm itself. This required security
must be reflected in the SMC technique’s adversarial model (i.e. what sort of privacy breaches it is intended to withstand).

Following the initial design, the designer must evaluate the performance of the PPDM algorithm to determine if it meets the application’s requirements. If not, the designer may determine that preserving the privacy of certain information which could be public is having too significant of an impact on the algorithm performance. Additionally, the designer may propose modifications to the original data mining algorithm itself which makes the PPDM version of it easier to implement (e.g. [JPW06]). The designer will then entire a cycle of implementing these changes and re-evaluating performance, finally halting when performance is acceptable. The remainder of this section discusses some of the aforementioned design criteria in greater detail, including partitioning, adversarial models, and privacy/performance tradeoffs.

Figure 4.1: A flowchart of a possible SMC-PPDM design process.
4.0.1 Partitioning

The partitioning, that is, how the input data is distributed among the parties, will largely restrict the design and efficiency of the potential algorithm. When records are horizontally partitioned, parties have different sets of complete records. When the records are vertically partitioned, the parties have different variables (database columns) for each of the records. The authors of [JW05] introduce “arbitrary partitioning”, in which records and columns are distributed arbitrarily to parties, with no necessary pattern. This subsumes the vertically and horizontally partitioned cases, and is the most general. Figure 4.2 illustrates the three types of partitioning.

Figure 4.2: Vertical, horizontal and arbitrary partitioning.
4.0.2 Adversarial Models

Adversarial models categorize the behaviors of the parties involved in the data mining process, to ensure the appropriate amount of security is enforced. Most work in privacy preserving data mining assumes the “semi-honest model,” as it is easier to work with. Semi-honest parties do not engage in malicious communication or hacking to disrupt the network, but instead they try to use the information that they have and the information received from other parties to find out as much as possible about other parties’ databases. The semi-honest model is not necessarily an unreasonable one: the data mining collaborations in question will typically be arrangements among companies or organizations, who are legally bound and accountable to not engage in malicious hacking, but would take advantage of any additional information received. One extension to the semi-honest model is to add accountability [JCK08]. That is, if a party breaks with protocol in such a way as to breach privacy, the party responsible can be identified.

Alternatively, one could use the “malicious” adversarial model, where other parties are allowed to perpetrate any attack in order to break the privacy. It should be noted that it was shown in [GMW87], that a protocol that is resistant to semi-honest adversaries can be made to resist malicious adversaries, albeit at the expense of significant computation and communication. One common concern among malicious parties is the potential for collusion. Parties that are willing to compare results of an intermediate SMC computation can potentially breach the privacy of the others. For instance, in the aforementioned secure sum [Sch95] with three parties, it is possible for two parties to
collude, and using their own values and the final sum, to find the value of the third party.

4.0.3 Privacy / Performance Tradeoffs

As a general principle, privacy and efficiency are inversely related. That is, the more privacy demanded by the application, the lower the performance of the algorithm will be. Ideal privacy is not always necessary [DZ02b]: for instance, in a lazy classifier where the classification is done on a per-query basis (e.g. the k-NN), the query may not necessarily need to be private. Take for instance, a medical application. If the query point to be classified represents an actual patient record to be evaluated for a disease, it may be important to preserve its privacy. However, if the same query were to be done with a hypothetical case for research purposes, the privacy of the query may not be important.

Another example of a privacy / performance tradeoff involves the sharing of class variables in classification problems. Some of the classifiers surveyed did maintain the privacy of the class labels (e.g. [XCL06, KC04b, VC04a]) while others did not (e.g. [ZCM05, YVJ06a, YJV06b]). The ease with which this is done can depend on numerous factors, including the partitioning and algorithm structure.

The importance of preserving the privacy of the class labels can depend largely on the domain. Suppose that in the example of a medical application, the class label is whether or not the patients in the training set have cancer. This information will almost certainly be restricted. If, for instance, the class label is something that can be obtained
from public record (e.g. their discretized lifespan for instance), there may be no need for preserving the class label privacy.
This chapter presents a suite of SMC-based privacy-preserving algorithms with different privacy-performance tradeoffs, implementing the Probabilistic Neural Network. The Probabilistic Neural Network (PNN) [Spe90] is an effective neural network architecture, underpinned by a theoretically sound framework. It can solve a variety of classification problems by approximating the Bayes optimal classifier. The Bayes optimal classifier is a theoretically optimal classifier which minimizes the expected misclassification rate. Because of the popularity and effectiveness of the PNN in the literature, designing and implementing a suite of privacy-preserving algorithms is useful in and of itself. But the algorithms are also constructed to demonstrate the complexities and difficulties that are currently associated with PPDM algorithm design. This analysis is leveraged later on in the design of a new framework to support PPDM algorithm implementation.

The PNN has been shown to be well suited for a variety of classification problems, and it’s regression counterpart, GRNN has been shown to be very effective on a variety of regression problems. In [KY03], the authors show that GRNN has the highest performance for detecting malignant breast cancers among all of the other algorithms tested in the paper. R2 Technology (http://www.r2tech.com) currently is utilizing the PNN algorithm in a hospital machine for detecting breast cancer. Therefore, despite the availability of many other machine learning approaches to solve classification problems (e.g.
multi-layer perceptron neural networks and support vector machines) the PNN maintains
its status as a highly accurate, well understood, and theoretically sound machine learning
algorithm. In this chapter, we show that PNN can be adapted to a privacy preserving en-
vironment and can offer many advantages. Besides its solid theoretical base and success
in practice, it is highly parallelizable, easy to implement, and in certain configurations,
lends itself to a high performance privacy preserving implementation. This is the first
privacy preserving version of the PNN to appear in the literature, and it is hoped that
this well established algorithm will find new uses in the domain of PPDM.

There are a number of practical scenarios where a privacy preserving implementation
of a data mining algorithm such as the PNN is desired. For instance, consider a scenario
where several hospitals want to use the data from their combined databases to train a
PNN classifier in detecting breast cancer. The consortium trying to accomplish this task
involves hospitals from several different regions, each one of which has essentially the
same information on different patients. This is a horizontal partitioning of the databases
containing the patient information. Hospitals could also, for instance, seek to merge
data for mining information from a shared set of patients, along with the data owned by
dietitians, medical testing centers, and research hospitals. This corresponds to vertical
partitioning of the data. The work presented in this paper will enable this consortium to
use the PNN as a predictor of future breast cancer instances by utilizing the data from
all the sites (independently of whether they are horizontally or vertically partitioned),
while at the same time preserving the privacy of the patient information for which each
organization cares.
The “training phase” of the PNN consists only of loading the training points. This lack of a training phase makes the PNN very well suited for on-line operation, when new data points are added to the training set. In PNN’s performance phase, in order for one to predict the label of a datum whose label is unknown (e.g., whether a new patient has breast cancer or not), some form of distance of this datum needs to be calculated to every data-point belonging to the training set.

Privacy preserving data mining often tries to simulate a situation where all data could be sent to a trusted third party and mined on a single system by that third party. For some mining tasks, e.g., mining of health care and criminal justice data, the transmission of this data to another party may violate privacy laws like HIPAA (the Health Insurance Portability and Accountability Act), the civil rights of the accused, and trade secrets. This is where privacy preserving data mining becomes most useful. Privacy preserving data mining (PPDM) borrows various techniques from disciplines like secure multiparty computation (SMC), among others, in order to mine the data from geographically distributed databases for useful conclusions, without disclosing too much information.

This is the first privacy preserving Probabilistic Neural Network presented in the literature. The paper presents a comprehensive family of four conceptually similar privacy preserving algorithms for the PNN. We analyze this within a framework of privacy/performance tradeoffs, and this analytical methodology can and should be applied to other privacy preserving data-mining algorithms. While most prior work in privacy preserving neural networks only separated the network and the owner of the data to be tested
[BOP06], these algorithms are trained and evaluated with data distributed among multiple parties. Also, we present results for actual implementations of the algorithms, which has been lacking in much of the SMC-based privacy preserving data mining literature (our PNN implementations are SMC-based implementations).

In table 5.1, we present a synopsis of other algorithms, compared to the family of privacy preserving PNN algorithms. PNN, like k-NN models belong to a class of “lazy” learning algorithms which are queried for every test point. Therefore, a consideration when surveying other privacy preserving data mining algorithms is whether or not the query is kept private. Algorithms that follow a query model are contrasted with other machine learning algorithms, like ID3, and SVM, whose output is a fully developed model. Therefore, the notion of a private query is not applicable to them. Of the three k-NN implementations surveyed, none offered the ability to use fully private queries, which will likely be necessary in many privacy preserving data mining applications. The PNN algorithms developed in this paper give the user the choice of keeping the query private while incurring greater computational expense. Additionally, only one of the vertically partitioned algorithms surveyed were able to keep the class labels of the training set private. All four of the PNN algorithms presented here preserve the privacy of the class labels. Assuming the class labels to be public is likely to be a stronger assumption than is practical in many cases. Finally, very few algorithms tested implementations, and the ones that did, did so with data sets that were relatively small (see table 5.1) for more comparisons. In our work, we are interested test the practicality of large scale
data mining, where one easily deals with tens of thousands of training data points and potentially larger data sets.

<table>
<thead>
<tr>
<th>Reference</th>
<th>Algorithm</th>
<th>Partitioning</th>
<th>Private Classes</th>
<th>Private Query</th>
<th>Size of Training Set Evaluated</th>
</tr>
</thead>
<tbody>
<tr>
<td>[XCL06]</td>
<td>k-NN</td>
<td>Horizontal</td>
<td>Yes</td>
<td>No</td>
<td>4177 pts, 29 dim.</td>
</tr>
<tr>
<td>[ZCM05]</td>
<td>k-NN</td>
<td>Vertical</td>
<td>No</td>
<td>Partially</td>
<td>None</td>
</tr>
<tr>
<td>[KC04b]</td>
<td>k-NN</td>
<td>Horizontal</td>
<td>Yes</td>
<td>No</td>
<td>None</td>
</tr>
<tr>
<td>[KV03]</td>
<td>Naive Bayes</td>
<td>Horizontal</td>
<td>Yes</td>
<td>N/A</td>
<td>None</td>
</tr>
<tr>
<td>[VC04a]</td>
<td>Naive Bayes</td>
<td>Vertical</td>
<td>Yes</td>
<td>N/A</td>
<td>None</td>
</tr>
<tr>
<td>[LP02]</td>
<td>ID3</td>
<td>Horizontal</td>
<td>Yes</td>
<td>N/A</td>
<td>None</td>
</tr>
<tr>
<td>[YVJ06a]</td>
<td>SVM</td>
<td>Vertical</td>
<td>No</td>
<td>N/A</td>
<td>958 pts., 27 dim.</td>
</tr>
<tr>
<td>[YJV06b]</td>
<td>SVM</td>
<td>Horizontal</td>
<td>No</td>
<td>N/A</td>
<td>958 pts., 27 dim.</td>
</tr>
<tr>
<td>PP-PNN</td>
<td>PNN</td>
<td>Horizontal</td>
<td>Yes</td>
<td>Yes</td>
<td>128,000 pts, 16 dim.</td>
</tr>
<tr>
<td>PP-PNN</td>
<td>PNN</td>
<td>Vertical</td>
<td>Yes</td>
<td>Yes</td>
<td>128,000 pts, 64 dim.</td>
</tr>
</tbody>
</table>

5.1 The PNN Algorithm

The PNN approximates the Bayesian Optimal Classifier. From Bayes’ Theorem, we have that the a-posteriori probability that an observed datum $x$ has come from class $c_j$ is given by the formula:

$$p(c_j|x) = \frac{p(x|c_j)p(c_j)}{p(x)}$$  \hspace{1cm} (5.1)

The Bayes classifiers chooses as the class that datum $x$ would have come from the class that maximizes this a-posteriori probability (this choice minimizes the misclassification error, resulting in a classifier that is optimal in the sense of minimizing this error). In
order to make effective use of Bayes’ formula, we must calculate the a-priori probabilities as the probability density functions of the datum $x$, given that it comes from class $c_j$ (i.e., $p(x|c_j)$). The a-priori probability can be estimated directly from the training data, that is, $p(c_j) = \frac{PT_j}{\sum_j PT_j}$ where $PT_j$ designates the number of points in the training data set that are of class $c_j$. The conditional probability density functions ($p(x|c_j)$) are calculated (see [Par62]), as follows:

$$p(x|c_j) = \frac{1}{(2\pi)^{D/2} \left( \prod_{i=1}^{D} \sigma_i \right)} \frac{PT_j}{\sum_{r=1}^{PT_j} \exp \left( -\sum_{i=1}^{D} \frac{(x(i) - X^j_r(i))^2}{2\sigma_i^2} \right) }$$  

(5.2)

where $D$ is the dimensionality of the input patterns (data), $PT_j$ represents the number of training patterns belonging to class $c_j$, $X^j_r$ denotes the r-th such pattern, $x$ is the input pattern to be classified, and $\sigma_i$ is the smoothing parameter along the $i$th dimension, used by the PNN classifier. The PNN algorithm identifies the input pattern as belonging to the class that maximizes the a-posteriori probability $p(c_j|x)$. It is assumed in the PNN, that all dimensions in training and testing data are normalized to the range of [0, 1]. This notation is summarized in appendix A.

The choice of the $\sigma$ parameter has an effect on PNN’s classification accuracy. In this paper, we assume that the $\sigma$ parameters have been appropriately chosen, and we will only be concerned in loading the training data into memory, prior to the initiation of the PNN’s performance phase. Once the loading is complete, a set of conditional probability densities (one for each class) is computed for each testing point (using equation (5.2)). The label of the testing point is then determined as the label of the class that maximizes the a-posteriori probability of equation (5.1) (ignoring $p(x)$, which is the same for all
classes). Figure 5.1 shows the neural network conceptualization of the PNN. Figure 5.2
does the pseudo-code of the PNN algorithm.

The pseudo-code is relatively easy to understand and implement. Beginning on line
1, the PNN iterates through every unique class $c_j$. It then loops through each member of
the training set $X_j$ which is in the current class $c_j$ (line 2). On line 3, it calculates sum of
the exponential of the distance between the testing point and the current training point.
This sum is a partial calculation of the class conditional probability density function
(CCPDF). In lines 4–5, it loops through and normalizes these calculations so that they
would accurately correspond to the conditional probability density functions, approxi-
mated by equation (5.2). Finally, on line 6, the class with the greatest class conditional
probability (CCP) as calculated by the CCPDF is selected as the class representing $x$.

5.2 Privacy Preserving Distributed PNN Algorithm

In this paper, we distinguish four different cases of a privacy preserving distributed PNN.
We make the distinction on both the type of partitioning (horizontal and vertical) as well
as the privacy of the query. Therefore, the four cases are: horizontal partitioning with
a public query, horizontal partitioning with a private query, vertical partitioning with a
public query, and vertical partitioning with a private query. In the literature, distinctions
are typically made only on the basis of partitioning. While it may seem unusual to make
a distinction on the privacy of the query, instead of assuming that the query is always
private, having a public query can save a significant amount of computation. In the
example of medical data mining, one could imagine that a private query could be used for the case of an actual diagnosis, where the query data would need to remain secret. In contrast, one could use a public query to pose a hypothetical or openly disclosed case to the system where there is no risk in exposing the query data.

First we state both the model of our analysis and our assumptions. The reader is referred to the appendix A of the paper where the notation, used throughout this chapter, is presented. Then we describe each of the four related algorithms in turn. The horizontally partitioned public query PNN works by performing the independent PNN computations on separate parties in parallel, finally combining the calculations with a simple and efficient secure sum at the end. The horizontally partitioned, private query
PNN works by using two rounds of a homomorphic-encryption-based secure scalar product. The vertically partitioned, public query algorithm works by finding the distances to each set of dimensions on the parties in parallel, and then combining these to yield the final calculation using both a secure sum and secure scalar product. The vertically partitioned, private query PNN works in a similar way to the public query case, except that it must use a homomorphic cryptosystem to preserve the privacy of the query.

5.2.1 Distributed Model and Assumptions

For the PP-PNN algorithm, we assume that we have at least 3 parties involved. The parties are “semi-honest”. That is, they do not engage in malicious communication or hacking to disrupt the network, but instead they try to use the information that they have and the information received from other parties to find out as much as possible about other parties’ databases.

A test point query can be issued by any of the participating parties. The party issuing the query is always referred to as a $P^q$. In the case of horizontal partitioning, the $D$-dimensional training data, $X_r$, in $S$, are divided among the parties, in some way, such
that each party $P^k$ owns several $D$-dimensional instances of the training data, with that set designated as $S^k$.

In the vertically partitioned case, each party $P^k$ owns one or more variables, for all of the available records. The set of variables owned by $P^k$ is denoted by $D^k$. The class labels are also assumed to be private. The party that owns them is designated as $P^c$. While in much of the literature, the class variables are assumed to be public, this may be a poor assumption. Suppose that in the example of a medical application, the class label is whether or not the patients in the training set have cancer. This information will almost certainly be restricted. We assume that there are no missing values, and that merging the databases from all of the different nodes would yield a complete set of variables and records.

5.2.2 Determining $\sigma$

While choosing a good $\sigma$ value for the PNN is important, it is outside the scope of the current research. It could be accomplished by evaluating the performance of various smoothing parameter values, $\sigma$, on a validation set. Depending on how exhaustive the search for parameters is, this could be a very computationally intensive process. However, in future work, it may be possible to adapt techniques like the one in [ZHM05], which presents a computationally inexpensive way to choose $\sigma$. 

35
5.2.3 Horizontally Partitioned Databases, Public Queries

The simplest privacy-preserving algorithm for PNN corresponds to the case where all of the data is completely horizontally partitioned and the query is public. In this case, we use an algorithm similar to the one presented in [SGM06] where a vector of conditional (on the class) probability density function values is passed around from node to node, with each node adding its calculated CCP’s to the vector. To begin with, all of the parties begin to calculate their respective portions of the final CCP vector. The final CCP can then be produced by using secure sum. This results in very minimal communication and also allows for significant parallelism within each party.

We assume the existence of a function called secureSum, capable of summing together a matrix across many parties, without any individual party knowing the intermediate terms, as in figure 3.2. This function, as used in the pseudo-code, takes three parameters. The first, is the party at which the secure sum will begin and end, the second is the share that each party is contributing to the sum, and the third is variable on the ending party where the final sum is to be stored.

When the query is public, the parties calculate individual class conditional probabilities (CCPs) and sum them. The pseudocode, shown in figure 5.3, is written in Single Program Multiple Data (SPMD) style, similar to an MPI program.

As seen in figure 5.3, similarly to the serial algorithm, the public query, horizontal algorithm must iterate over every class. First it calculates a secure sum to determine the number of points belonging to a particular class; we assume that this final result is
foreach $c_j$ do
  secureSum($P^q$, $PT^k_j$, $PT_j$);
  foreach $X^{k,j}$ do
    $CCP^k_j += \exp \left( -\sum_{i=1}^{D} (x(i) - X^{k,j}(i))^2/(2\sigma^2_i) \right)$;
  secureSum($P^q$, $CCP^k_j$, $CCP_j$);
if $k == q$ then
  foreach $CCP_j$ do
    $CCP_j /= (2\pi)^{D/2} PT_j \prod_{i=1}^{D} \sigma_i$;
  $C(x) = \arg\max_j \{CCP_j (PT_j/PT)\}$;

Figure 5.3: The PP-PNN algorithm for the horizontally partitioned public query case

a piece of public knowledge. However, if it is not, the sum can be easily shared between two parties, and securely computed using Yao circuits. Because this is relatively small computation, this could be achieved at the cost of little extra computation and communication. However, assuming that this piece of information is public greatly simplifies the algorithm. The secure sum that calculates the class probabilities occurs in line 2 of the code. In lines 3–4, at every part, a partial conditional (on every class) probability density function value is calculated. Because each conditional probability density function value is proportional to the sum of the exponentials of the distances between the query point and the training points, this can be summed together at every party individually, and then summed together again among parties. This is exactly what happens on line 5, with the final sum ending up at $P^q$. Lines 6–9 are executed at $P^q$, and are identical to the calculations performed for the serial algorithm.

A detailed example of the horizontally-partitioned, public-query PNN can be found in appendix section B.4.
5.2.3.1 Performance Analysis

There is relatively minimal communication required for this version of the PP-PNN algorithm. The first round of communication needed to determine the number of members in each class requires that a vector of size $J$ needs to be communicated between all parties resulting in $O(JK)$ time for both communication and computation (line 2). Thereafter, each party must find the squared difference between the components of all $X^{k,j}_i$’s and the test point $x$, take the exponential of them, and sum them together (lines 3–4). This is similar to what needs to be done with the serial PNN algorithm, and therefore takes $O(D \max PT^k)$ computation time, as it can be done in parallel. Each party must use the secure matrix summation to add the partial class conditional probability vectors together (line 5). Again, this requires $O(JK)$ time for both communication and computation because the intermediate sum must pass from party to party. Finally, the last node must search through the CCP vector to find the class with the largest probability, needing $O(J)$ time for computation (lines 6–9). Therefore, this PP-PPN version requires $O(JK + D \max PT^k)$ time for computation and $O(JK)$ for communication. This compares to $O(J + D(PT))$ operations for standard serial PNN. In this case, the PP-PNN algorithm takes advantage of the parallelism allowed by the distributed data.

5.2.3.2 Security Analysis

For this algorithm to be secure, intermediate results obtained by any party must meet the SMC definition of security. By the SMC definition of security, no party must be able
to glean any information that has not either been declared public or that it could not recover by using its own data and the result of the computation.

**Theorem 1.** The horizontally partitioned, public query PNN algorithm given in figure 5.3 is secure by the SMC definition of security.

**Proof.** There are only two portions of the algorithm which require communication among parties, and therefore only two opportunities for privacy to be breached (lines 2 and 5). Because the secure sum has been proven to be secure with three or more parties [Sch95], the secure addition of the $PT$ vector in line 2 will not reveal anything other than the overall vector of class frequencies ($PT_j$), which has been declared public. While this too can be kept private, it is typically an acceptable quantity to reveal, and revealing it allows the algorithms to be simplified.

The secure sum on line 5 only exposes the unnormalized CCP to $P^q$. The CCP that ends up at $P^q$ is an intermediate value to the computation, not the final value. However, the final value of the computation is the CCP vector, divided by the factor in line 8. Because party $P^q$ could glean this information from the final result along with its own information, nothing is unduly revealed under the SMC definition of security. All other parties have only calculated their own partial CCP vectors, or have been party to a secure sum where they cannot determine any other party’s CCP values, by definition of the secure sum.

By the composition theorem (see [Gol98]) *if all components of the protocol are secure then the protocol itself is secure.* Therefore, this algorithm is secure.
5.2.4 Horizontally Partitioned Databases, Private Queries

To keep the query private, the party that is performing the query must interact with every other party, computing the distance between \( x \) and each \( X_{r}^{k,j} \), but doing so in a way that does not reveal the distance to either party. Revealing the full distance could easily allow one of the parties to solve for the query point or allow the querying party to solve for some of the other data points. Therefore, we employ secret sharing in a way similar to [GLL04]. The result is shared between the parties in such a way that neither party has any usable information about the result, but together, the parties can produce the result. If we can randomly share the result between the parties, each party can then calculate its own CCP, and securely sum them together in a way that is similar to how the public query version of the algorithm operates.

To securely find the distance between \( x \) and each \( X_{r}^{k,j} \), we use as our primary component the secure scalar product protocol from [GLL04]. \( P^A \) generates a public/private key pair, encrypts all the elements of \( x^A \) and sends them to \( P^B \). \( P^B \) puts each encrypted element of \( cx(i) \) to the power of the corresponding element of \( x^B(i) \), and then takes the product of all of these numbers. Because of the properties of the homomorphic encryption system, this is equivalent to multiplying the corresponding elements together and adding those products. However, while \( P^B \) performs this computation, it cannot decrypt this value, because only \( P^A \) has the private key. Despite this, \( P^B \) does subtract a random share \( s^B \), which only \( P^B \) knows. \( P^B \) sends the encrypted remainder back to \( P^A \) where \( P^A \) can only decrypt the value of the scalar product minus the random share that \( P^B \)
owns. Therefore, neither party knows the full value of the scalar product, but each party has a random share of it.

To more easily look at the private query, PP-PNN algorithm in terms of the secure scalar product, let us first define normalized versions of each vector.

\[ x'(i) = \frac{x(i)}{\sqrt{2\sigma_i}} \forall i = 1...D, \text{ and} \]

\[ X'_r = \frac{X'_r(i)}{\sqrt{2\sigma_i}} \forall i = 1...D. \]

Now, we can rewrite the class conditional probability equation as:

\[
p(x|c_j) = \frac{1}{(2\pi)^{D/2} \left( \prod_{i=1}^{D} \sigma_i \right) PT_j} \sum_{r=1}^{PT_j} \exp(-dis^2(x', X'_r)) \quad (5.3)
\]

where \( dis(x', X'_r) \) gives the Euclidean distance between the query (testing point) and the corresponding training point. At this point, we can write the distance in terms a scalar product and two summations, as follows:

\[
dis^2(x', X'_r) = \sum_{i=1}^{D} (x'(i))^2 + \sum_{i=1}^{D} (X'_r(i))^2 - 2x' \cdot X'_r \quad (5.4)
\]

As pointed out previously in [HH06], a secure scalar product can be utilized for this problem.

The algorithm for the private query, horizontally partitioned PNN is presented in figure 5.4. The private query algorithm begins with a secure sum in lines 1–2, again used to calculate the number of training points in each class. This result is made public to all of the parties. After this, the negative squared Euclidean distance between the query...
point $\mathbf{x}'$ and every training point $\mathbf{X}'^{k,j}_r$ must be calculated. As shown in equation (5.4), to calculate this distance, we must have the sum of the squared components of both, $\mathbf{x}'$ and $\mathbf{X}'^{k,j}_r$. This is easy for the respective parties to obtain. To complete the distance calculation, the scalar product between $\mathbf{x}'$ and $\mathbf{X}'^{k,j}_r$ is needed. Lines 3–8 are executed on the querying party, and lines 9–17 are executed on every other party. On line 4, $P^q$ starts iterating over the set of all parties, because it must communicate with each of them to calculate the necessary distances. Each party must iterate through each class $c_j$. For every training point in every class, $P^q$ must engage in a scalar product calculation between the query (testing) point $\mathbf{x}'$ and every training data-point $\mathbf{X}'^{k,j}_r$ residing at party $P^k$. However, party $P^q$ does not know to which classes a party’s points belong. Only each individual party $P^k$ knows the class label of its own points. However, this is easily solved.

As the algorithm enters line 6 on $P^q$ and line 11 on the other parties, they are performing a shared scalar product over the non-querying parties’ training points. However, there is a modification to the standard secure scalar product. Because each party does not want to reveal what training points are members of what classes, or how many of each class that the party has, the party must generate a spurious value for the share of the secure scalar product when the training point is not in the current class in question. The non-querying parties keep track of what shares are valid and which are not, and will filter them out later in the algorithm. Therefore, in line 13, the party will calculate the share of a particular training point properly if it owns that training point in $\mathbf{X}'^{k,j}_r$; if it does not, it will randomly make a share from a random distribution that looks the same
as the real values (lines 15–16). Each party $P^k$ adds these spurious random distance shares to its distance shares, so that it has exactly $PT_j$ shares in each class and therefore the class labels of its training points as well as their number are not revealed to $P^q$. On line 14–16, the parties $P^k$ make and send these spurious random shares. Actual secure scalar products are performed at the non-querying parties on lines 12–13. On line 7, party $P^q$ engages in a secure scalar product not knowing whether it is between real or spurious values. In this pseudo-code, the $\text{sharedSSP}$ (shared secure scalar product) is the function presented in figure 3.3. It is meant to be called by each party, with the parameter being the vector owned by their respective parties. On each party, it will return the respective random shares $s^A$ and $s^B$ to $P^q$ and the non-querying parties respectively. The respective random shares can then be added to the respective sum of squared components (again, line 7 and line 13). Note that while in the pseudo-code each secure scalar product is listed as a separate operation, in practice, the transfers for the secure scalar product should be done in bulk, to avoid repeated communication latency between each party.

Each party $P^k$ now owns a random share of the distance between each of its training points $X_{r^{k,j}}$ and the test point $x'$. All of the non-querying parties also know which shares are spurious. All parties then perform the exponential of their respective shares. Finally, a scalar product is performed on the exponential of these shares, sharing the results of this between $P^q$ and each $P^k$ (lines 8 and 16 for $P^q$ and the non-querying parties respectively). On line 8, $P^q$ also adds its own distance calculations to the share of the CCP (it is faster because it does not have to encrypt these distances). This scalar product skips operating
on the spurious shares on the non-querying parties, because they are aware which values
are spurious. The secure summation of all of these will then yield the CCP vector to
produce the final result (lines 18–19). Finally, at $P^q$, the class conditional probabilities
are divided by the appropriate factors and a label is found through linear search (lines
20-23).

```plaintext
foreach $c_j$ do
  secureSum($P^q$, $PT^k_j$, $PT_j$);
if $k == q$ then
  foreach $P^k$ do
    foreach $c_j$ do
      foreach $r = 1...PT_j$ do
        $s_A^r = \exp(2 \cdot \text{sharedSSP}(x') - \sum_{i=1}^D (x'(i))^2)$;
        $CCP^k_j += \text{sharedSSP}(s_A) + \sum_{r^q,j} -\text{dis}^2(x', X^q_{r,j})$;
  else
    foreach $c_j$ do
      foreach $r = 1...PT_j$ do
        if $X'^j_r \in X'^{k,j}_{r'}$ then
          $s_B^r = \exp(2 \cdot \text{sharedSSP}(X'^{k,j}_{r'}) - \sum_{i=1}^D (X'^{k,j}_{r'}(i))^2)$;
        else
          $s_B^r = \exp(\text{randfloat})$;
          send($P^q,$randfloat$)$;
          $CCP^k_j += \text{sharedSSP}(s_B)$;
  endforeach
  secureSum($P^q$, $CCP^k_j$, $CCP_j$);
else
  foreach $CCP^k_j$ do
    secureSum($P^q$, $CCP^k_j$, $CCP_j$);
if $k == q$ then
  foreach $CCP_j$ do
    $CCP_j /= (2\pi)^D/2 PT_j \prod_{i=1}^D \sigma_i$;
    $C(x) = \text{argmax}_j \{CCP_j(PT_j/PT)\}$;

Figure 5.4: The PP-PNN Algorithm for the horizontally partitioned private query case

A detailed example of the horizontally-partitioned, private-query PNN can be found
in appendix section B.5.

44
5.2.4.1 Performance Analysis

To begin with (lines 1–2), this algorithm requires $O(JK)$ for both computation and communication to complete the secure sum. Each $P^k$ must also calculate the summation of the components of each of its $X^{k,j}_r$ (lines 7 and 13). However, these calculations can proceed in parallel, so this adds $O(D \max PT^k)$ operations to the process.

In lines 7 and 13–16, the most intensive part of the algorithm is entered, where a secure scalar product is done over each training point. Each party $P_k$ adds spurious values to the vectors it returns, expanding its number of values from $O(PT^k)$ to $O(PT)$ (lines 14–16). Communication for this portion can be optimized by only sending the $D$-dimensional operand $x^\prime$ over once for all $K$ parties. Party $P^q$ must also receive each of $PT$ shares from each party, yielding $O(KD + KPT)$ for the communication in this portion (with the $KD$ portion from the transmitted query). There are $O(KED(\max PT^k))$ computations for this step over all parties, where $E$ represents encrypted operations. Because of the stark contrast in performance between standard arithmetic and the encrypted arithmetic that must be done in this algorithm, a new constant $E$ is introduced to take into account the time required by an encrypted operation, compared to $O(1)$ for the corresponding standard operation.

A scalar product must be computed between every party in order to get the CCP shares, thus requiring $O(EK(PT))$ computations overall. Because each encrypted share must be sent over, it requires $O(KPT)$ communication for this part. Finally, these CCPs must be securely summed together, taking $O(JK)$ computation and communication. Consequently, the final computational of a private query PP-PNN algorithm ends up
being equal to \( O(KED(\max PT^k) + EK(PT) + D\max PT^k + JK) \). The communication complexity is equal to \( O(JK + KD + KPT) \). The performance of this algorithm is slower compared to the public query case. The additional time needed by this version of the PP-PPN algorithm is mostly needed to compute the secure scalar products.

### 5.2.4.2 Security Analysis

In this section, the security of the horizontally partitioned, private query PNN is proven.

**Theorem 2.** The horizontally partitioned, private query PNN algorithm given in figure 5.4 is secure by the SMC definition of security.

**Proof.** During the secure sum in lines 1–2, \( P^q \) receives the total number of training points in each class, which as before is considered acceptable public knowledge. By prior security proof of the secure sum [Sch95], it also known that the secure sum computation itself conforms to the SMC security definition.

By lemma 3, lines 3–7 on \( P^q \) and lines 9–16 on all other parties are secure. The secure scalar product which disregards spurious shares, taking place on lines 8 and 17 for \( P^q \) and \( P^k \) respectively, is secure by lemma 4.

In lines 18 and 19, \( P^q \) receives the unnormalized \( CCP_j \) values, which can be generated from the final probabilities and \( PT_j \), which is public knowledge. Therefore, this revelation is also secure by the SMC definition.

Again, by the composition theorem, because all sections of this algorithm are private with respect to the SMC definition, the algorithm itself is secure.
Lemma 3. The actions performed in lines 3–7 and lines 9–16 in the horizontally partitioned, private query PNN are secure by the SMC definition of security.

Proof. This lemma and others depend on a proof by simulation. Proof by simulation states that if each party can simulate each part of the algorithm through using its own knowledge and public knowledge, then the algorithm is secure. The proof can be further simplified by concentrating only on simulating the messages received, as each party’s local calculations and messages sent can be generated from the local knowledge and the received messages.

On line 7, \( P^q \) receives a share of a secure scalar product or a spurious value. Because the existing security proofs for the secure scalar product [GLL04] emphasize that no additional knowledge about the total is gained from this share, \( P^q \) can simulate these with random numbers drawn from the same field. The same process can simulate the spurious values. In line 13, the data received from \( P^q \) during the secure scalar product is entirely encrypted and indecipherable by the other parties and can be simulated using random encrypted numbers.

Lemma 4. The secure scalar product performed on lines 8 and 17 of the horizontally partitioned, private query PNN are secure by the SMC definition of security

Proof. The secure scalar product performed between \( P^q \) and each \( P^k \) on lines 8 and 17 of figure 5.4 is similar to the one in [GLL04]; the difference is that, because \( P^k \) knows which of those shares are spurious, it knows not to include them in the final result. A proof by simulation will show that this secure scalar product reveals nothing additional.
To begin with, $P^q$ transmits $PT$ encrypted distances to $P^k$. Because the size of the vector $PT$ is public knowledge, and the distances are encrypted with only $P^q$ owning the private key, $P^k$ can simulate this message by generating $PT$ random encrypted numbers.

The homomorphically-encrypted operations are then performed on $P^k$. The number of valid shares owned by $P^k$ can be in the range of $[0, PT]$. Therefore, the range of the resultant share ($[PTe^{-D}, PT]$) is the same for every party $P^k$, and is well known by all parties. $P^q$ can simulate the final share it receives by simply drawing a random number from that same range. Because all information received can be simulated, this portion of the algorithm is secure.

5.2.5 Vertically Partitioned Databases, Public Queries

When the databases are vertically partitioned across the parties, we face a different set of difficulties. As stated before, we do not assume that the classes are public knowledge, and therefore, this column of classes must be owned by one of the parties, which we refer to as $P^c$. For all parties, the set $D^k$ gives all of the vertical partitions or columns owned by $P^k$ (as opposed to $S^k$ for the horizontal partitions in the horizontal case). It is assumed that every party has the complete column over every record. The vertically partitioned, public query case is presented in figure 5.6.

Lines 1–2, generate the public key and private key on the querying party. The querying party broadcasts its public key and its unencrypted query on lines 3–4. On lines 5–7, a secure sum calculates each $PT_j$ and the result is broadcast. We are considering these
if $k = q$ then
  $(sk, pk) = \text{GenerateKeyPair}();$
  broadcast($pk$);
  broadcast($x'$);
  foreach $c_j$ do
    secureSum($P^q, PT^k_j, PT_j$);
    broadcast($PT_j$);
  endforeach
  foreach $X'^{k,j}$ do
    $Y^k_r = \sum_{i \in D^k} (X'^{k,j}_r(D^k(i)) - x'(i))^2$;
    if $c = q$ then
      sharedSecureSum($P^q, P^g, Y^k_r, Y_r$);
    else
      sharedSecureSum($P^q, P^c, Y^k_r, Y_r$);
    endif
  endforeach
  if $c = q$ and ($k = q$ or $k = g$) then
    foreach $c_j$ do
      if $k = q$ then
        $CCP_j = \text{SSP}(\text{pad}(\exp(-Y_r), c_j))$;
      else if $k = g$ then
        $\text{SSP}(\exp(-Y_r))$;
      else
        $\text{SSP}(\exp(-Y_r))$;
      endif
    endforeach
  else
    foreach $c_j$ do
      if $k = q$ then
        $CCP_j = \text{SSP}(\exp(-Y_r))$;
      else if $k = c$ then
        $\text{SSP}(\exp(-Y_r))$;
      endif
    endforeach
  endif
  if $k = q$ then
    foreach $CCP_j$ do
      $CCP_j /= (2\pi)^{D/2} PT_j \prod_{i=1}^{D} \sigma_i$ ;
      $C(x) = \arg\max_j \{CCP_j (PT_j / PT) \}$ ;
  endforeach
end if

Figure 5.5: The PP-PNN Algorithm for the vertically partitioned public query case
final $PT_j$ values to be public. Then, in lines 8–9, all parties iterate over their training points $X^{k,j}_r$, calculating $\sum_{i \in D^k} (X^{k,j}_r(D^k(i)) - x(i))^2$ for each training point. This is kept in the array $Y^k_r$.

The lines 10–13 then add these $Y^k_r$ arrays using a secure sum, and sharing that secure sum between different parties. Here, the function $sharedSecureSum$, discussed earlier, is introduced. It is similar to the previously presented secure sum algorithm except that instead of returning the full sum to the final party, it shares the sum between two parties. It takes four parameters. The first two are the two parties that will share the sum. The third parameter is the partial value that will be added in the sum, and the fourth parameter is the variable where the sum shares will end up on the two appropriate parties.

There are two options with sharing this sum. If the $q \neq c$, that is, if the querying party and the party with the classification are different, it is most efficient to share the distances between them (lines 12–13). If the party originating the query and owning the classes are the same, $q = c$, then the distances are shared with another party, $P^q$ which is randomly selected, but is neither $q$ nor $c$ (lines 10–11). If this were to remain unshared between different parties, then $P^q$ would be able to fully decrypt all of the distance values, violating privacy.

Now, the requisite distances have been calculated for each point in $X^{k,j}_r$, and they are shared between two parties, with each share in the variable $Y_r$ on the two parties. The sum of the exponential of these distances must now be calculated to yield the class conditional probabilities. This is done in lines 14–25. In this pseudo-code, it is assumed
that functions used apply to each element of the vector. Again, different paths are chosen if the querying party and the party holding the classes are the same party. At line 14, it is checked if $c = q$. If so, in lines 16–19 both $P^c$ and $P^q$ engage in a scalar product. This operation, $SSP$, is not a shared scalar product but gives $P^q$ the entire answer. This is a trivial change to the previously presented secure scalar product with the exception that a share is not subtracted. In this case, if the results of the Euclidean distance secure computations were to end up at $P^c$, they would also end up at $P^q$, which would be able to decrypt the distances from its query point and glean additional information, therefore violating privacy.

If $P^q$ and $P^c$ are not the same party, as in lines 20–25, then the class conditional probabilities are relatively easy (and more efficient) to calculate. Each party takes the negative exponential of their respective shares of the calculated distance values. Then, iterating over each class, a secure scalar product is calculated and given back to $P^q$. This secure scalar product takes place only between corresponding shares in the same class. Through this secure scalar product function, $P^c$ receives all of the shares of $s^A_c$ from $P^q$, discriminating between valid and invalid shares. In lines 14–19, we must use a different approach. For each class $c_j$, $P^c$ then makes a vector of size $PT$. For training points that are members of the particular class $c_j$, their shares are placed into the respective positions. Zeros are placed in all other positions by the pad function. The pad function takes, two parameters. The first is the vector to pad, and the second is the class to preserve (not pad with zeros). Then the secure scalar product is executed. In this, the padded vector is then encrypted and sent to $P^g$. $P^g$ then performs a secure scalar product between
this vector and its own share vector. It sends the final result back to $P^q$, where $P^q$ can decrypt it. The scalar product is calculated between the proper shares because the zeros placed at all other positions preclude the respective elements from being included in the scalar product. In lines 26–28, $P^q$ divides the CCP values it has received by the factors necessary to correct the calculation. Finally, in line 29, the class with the largest class conditional probability is selected for a testing point.

A detailed example of the vertically-partitioned, public-query PNN can be found in appendix section B.6.

### 5.2.5.1 Performance Analysis

The performance of the vertically partitioned public query case is as follows. The first secure sum is $O(JK)$ for computation and communication. Broadcasting the query and the test point are relatively minor in comparison and are therefore ignored. Initially, the square sum of the individual party’s training points must be calculated, giving $O(PT_{max}|D^k|)$ for the overall operation, because the parties do this in parallel (lines 8–9). The shared secure sum takes $O(KPT)$ (lines 10–13) for both computation and communication. The secure scalar product is then $O(E(PT))$ computations per party (with the additional $E$ term representing the time for encrypted arithmetic) and must be done with each of the $K$ parties serially, resulting in $O(EK(PT))$ computation (lines 14–25). This also results in $O(KPT)$ communication. Finally, there is a search that takes
$O(J)$ time to give the final result. Overall, this requires $O(JK + PT_{\max}|D^k| + EK(PT))$ time for computation (lines 26–29), as well as $O(JK + KPT)$ total communication.

5.2.5.2 Security Analysis

In this section, the security of the vertically partitioned, public-query PNN is proven.

Theorem 5. The vertically partitioned, public-query PNN algorithm given in figure 5.5 is secure by the SMC definition of security.

Proof. In lines 3–4, all parties besides $P_q$ receive both the public key and the test point query, which are considered public. The secure sum of $PT_j$ values (lines 6–7) has already been proven secure. The result of that computation, the $PT_j$ values received by all parties, are acceptably revealed because these overall quantities have been declared public.

For lines 10–13, there are two different cases, with two possible secure sums. As is shown in lemma 6, the final shares reveal no additional information, and therefore, these lines are secure. Lemma 7 furthermore shows that lines 14–25 are secure.

By the composition theorem, because all of the portions of the algorithm are secure, the overall algorithm is also secure. □

Lemma 6. The function sharedSecureSum is secure by the SMC definition of security.

Proof. At the beginning of the algorithm, before each share of the vector $Y^k_r$ is split, it is known that the sum $\sum_k Y^k_r = Y_r$ is in the range $0 \leq Y_r \leq D$. Splitting each element into two nonnegative shares is equivalent to splitting the final sum into two nonnegative
shares. Because the split is done randomly, each share, we will call them \( A(Y_r) \) and \( B(Y_r) \), are in the range \( 0 \leq A(Y_r), B(Y_r) \leq D \). Therefore, the shares which will be used in each of the independent secure sums, conform to the constraints of the non-shared secure sum.

The shared secure sum is therefore secure by the composition theorem \cite{Gol98}, because it is composed of two secure sums, which have already been shown secure.

\[ \square \]

**Lemma 7.** The actions in lines 14–25 of the vertically partitioned, public-query PNN algorithm in figure 5.5 are secure

**Proof.** In lines 14–19, the are two sets of messages received. In participating a secure scalar product, party \( P^q \) receives the quantity in \( CCP_j \) (line 17) which is only different from the final answer by a publicly known quantity (the divisor in line 28). Therefore by using this final answer and public information, it can effectively simulate receiving this quantity. In participating in the secure scalar product, party \( P^g \) receives only encrypted quantities, which it cannot decrypt (line 19) and can therefore simulate by random strings.

In lines 20–25, a similar situation is encountered, wherein \( P^q \) as before receives something different from the final answer only by factor of the divisor in line 28: therefore \( P^q \) can easily simulate that message using that publicly known divisor and the final answer. Additionally, party \( P^c \) receives only encrypted strings during the secure scalar product, which it can easily simulate. Therefore, by simulation, these lines are secure. \[ \square \]
5.2.6 Vertically Partitioned Databases, Private Queries

In the vertically partitioned, private query case, the algorithm is similar to the public query case. It is more computationally demanding, because it involves extra homomorphically encrypted multiplication and addition. The algorithm is as follows.

As before, in lines 1–3, the public and private key are generated by the querying party and the public key is broadcast to all of the parties. In line 4, the encrypted query is broadcast. Again, in lines 5–7, the $PT_j$ values are calculated, and broadcast.

Lines 8–11 pertain to calculating the square of the Euclidean distance. Recall that the distance is:

$$ \text{dis}^2(x', X'_j) = \sum_{i=1}^{D} (x'(i))^2 + \sum_{i=1}^{D} (X'_j(i))^2 - 2x' \cdot X'_j $$

In lines 8–9, every party calculates the sum of the squares of the distance components over each of its available dimensions (recall the operator $\prod$ is repeatedly applying the operation $\otimes$). Specifically, in the private query version of the vertically partitioned algorithm, the negative distance is calculated, because of the limitations of the Paillier cryptosystem (problems multiplying by a -2). To complete the distance sum, $P^q$ must add the encrypted sum of the query point $x'$ on lines 10–11.

In lines 12–15, a shared encrypted secure sum is performed. The function $\text{sharedEncryptedSecureSum}$, is a tree structured operation for efficiency. It takes four parameters. The first two are the parties that will share the final plaintext value. The third parameter is the encrypted operand to involve and the final parameter is where the result will
if $k = q$ then
1 \[(sk, pk) = \text{GenerateKeyPair}();\]
broadcast(pk);
broadcast(\text{Enc}(x'));
foreach $c_j$ do
\[
\text{secureSum}(P^q, PT^k_j, PT_j);
\]
broadcast(PT_j);
foreach $X'^{k,j}(D^k(i))$ do
\[
\text{Enc}(Y^k) = \text{Enc}\left(\sum_{i \in D^k} - (X'^{k,j}(i))^2\right) \otimes \left(\prod_{i \in D^k} \text{Enc}(x'(i))X'^{k,j}(i)^2\right);
\]
if $k = q$ then
\[
\text{Enc}(Y^k) = \text{Enc}(Y^k) \otimes \text{Enc}(\sum_{i \in D^k} -(x'(i))^2);
\]
if $c = q$ then
\[
\text{sharedEncryptedSecureSum}(P^q, P^c, \text{Enc}(Y^k), Y_r);
\]
else
\[
\text{sharedEncryptedSecureSum}(P^q, P^c, \text{Enc}(Y^k), Y_r);
\]
if $c = q$ and $(k = q$ or $k = g)$ then
\[
\text{foreach } c_j \text{ do}
\]
if $k = q$ then
\[
\text{CCP}_j = \text{SSP}(\text{pad}(\text{exp}(Y_r, c_j)));
\]
else if $k = g$ then
\[
\text{SSP}(\text{exp}(Y_r));
\]
else
\[
\text{foreach } c_j \text{ do}
\]
if $k = q$ then
\[
\text{CCP}_j = \text{SSP}(\text{exp}(Y_r));
\]
else if $k = c$ then
\[
\text{SSP}(\text{exp}(Y_r));
\]
if $k = q$ then
\[
\text{foreach } \text{CCP}_j \text{ do}
\]
\[
\text{CCP}_j /= (2\pi)^D/2PT_j \prod_{i=1}^D \sigma_i ;
\]
\[
C(x) = \argmax_j \{\text{CCP}_j(PT_j/PT)\} ;
\]
\[
\text{Figure 5.6: The PP-PNN Algorithm for the vertically partitioned private query case}\]
be stored. Therefore, the operation ⊗ in the figure 3.5 is replaced with large encrypted operation specified in [Pai99]. Again, as in the public query case, we must make a distinction between the cases where \( q = c \). If \( q = c \), then the result of the secure sum must be shared with \( P^9 \) instead. From this point on (lines 16–31), the algorithm is identical to the public query case with the exception that the values of \( Y_r \) on lines 19, 21, 25 and 27 are not negated.

One important constraint that we add to the original paradigm is for our homomorphically encrypted operations. We stipulate that the host with the private key \( pk \), remain a leaf of the tree. On any other part of the tree, it could potentially decode part of the final operation, more so than the SMC definition or security would allow. If, for instance, a party with the public key were on the second level of the tree, it would be able to completely decrypt information received from a leaf node. So, without loss of generality, we assume that the party with the private key is at a leaf node, and that result of the computation must end up at party \( P^1 \). We can temporarily renumber the parties if this is not the case.

Again looking at the pseudo-code for the SMC tree-based operations in 3.5, if the party executing the code is \( P^1 \), as in lines 1–2, then it merely applies the operation \( \otimes \) to what it receives from \( P^2 \) and \( P^3 \). Optionally, a share or the entirety of this final result can be sent back to the party with the private key (not shown in pseudocode). Lines 3–4 are executed by the parties which are leaf nodes in the tree. In this, they merely send their values to their parent nodes. We assume that value \( v^k \) has been appropriately processed for privacy. In the case of homomorphic operations, it would have been encrypted. Lines
5–6 are executed by all other parties, where they receive the processed operands from their children (again, appropriately processed) and apply $\otimes$ to those received operands along with its own. Finally, it sends the result to its parent party.

A detailed example of the vertically-partitioned, private-query PNN can be found in appendix section B.7.

5.2.6.1 Performance Analysis

The private query case has similar performance characteristics to the public query case but with some additional encrypted operations. The initial secure sum to determine the number of points in each class takes $O(JK)$ for both computation and communication. The square sum of the individual party’s training points must be calculated, giving $O(PT_{max}|D^k|)$ (lines 8–9) computation for the overall operation, because the parties do this in parallel. Also in parallel, the parties are computing an encrypted scalar product (line 9), overall requiring $O(E(PT)_{max}|D^k|)$ computation time ($E$ taking the time of encrypted operations into account). The shared encrypted secure sum takes $O(E(PT)\log(K))$ computation and $O(PT\log(K))$ communication (lines 12–15), because it makes use of a tree structure. The secure scalar product is then an $O(E(PT))$ computation and $O(PT)$ per party, and must be done with each of the $K$ parties serially, resulting in $O(EK(PT))$ time computation and $O(KPT)$ time communication (lines 16–27). Finally, there is a search that takes $O(J)$ computation to give the final result.
(lines 28–31). Overall, this requires $O(JK + E(PT)(\max|D_k| + \log(K)) + EK(PT))$ computation and $O(JK + PT\log(K) + KPT)$ for communication.

### 5.2.6.2 Security Analysis

In this section, the security of the vertically partitioned, private-query PNN is proven.

**Theorem 8.** The vertically partitioned, private-query PNN algorithm given in figure 5.6 is secure by the SMC definition of security.

**Proof.** On line 3, all of the parties received the public key, which is of course, public knowledge. The parties also receive the encrypted query. $P^q$ already knows this query, and by definition of the cryptosystem, it is unable to be understood by the other parties.

Again, it is known by the secure sum proof that the secure sum on line 6 is secure, and its result, the quantities $PT_j$ which are broadcast, are considered public knowledge.

Lines 10 and 11 involve only local operations, and because of the homomorphic cryptosystem used, do not reveal additional information. According to lemma 9, lines 12–15 of the algorithm are secure. The security of lines 16–31 were already proven for the vertically partitioned, public query algorithm (lemma 7) and are omitted for brevity.

By the composition theorem, because all sections of the algorithm are secure, the overall algorithm is secure.

**Lemma 9.** The actions in lines 12–15 of the vertically partitioned, private-query PNN algorithm in figure 5.6 are secure.
Proof. This lemma will be proven by simulation. During the shared encrypted secure sum, only parties without the private key receive messages, because the only party with the private key, \( P^q \) is a leaf node of the secure tree, and therefore receives no message during the tree-based communication. All other parties receive only encrypted operands during this phase, which can be easily simulated by random strings.

Because random shares of the final sum are split between either \( P^q \) and \( P^g \) or alternatively between \( P^q \) and \( P^c \), neither party is able to obtain any additional information about the final sums. Therefore, because all parties know the limits \([e^{-D}, D]\) of the sum, both parties can simulate their respective shares with a vector of random numbers chosen from the same range.

5.2.7 Algorithm Summary

Table 5.2 summarizes the computation and communication requirements for all of the secure PNN algorithms. To recap the notation, \( PT \) is the number of points in the total training set, \( PT^k \) is the number of points at party \( P^k \), \( K \) is the total number of parties, \( D \) is the number of dimensions, \( J \) is the number of classes in the classification problem, and \( E \) is a multiplier signifying more expensive encrypted operations.
Table 5.2: Comparison of PP-PNN Algorithms

<table>
<thead>
<tr>
<th>Partitioning</th>
<th>Private Query</th>
<th>Computation</th>
<th>Communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horizontal</td>
<td>No</td>
<td>$O(JK + D_{max}PT^k)$</td>
<td>$O(JK)$</td>
</tr>
<tr>
<td>Horizontal</td>
<td>Yes</td>
<td>$O(KED(\max PT^k) + EK(PT) + D_{max}PT^k + JK)$</td>
<td>$O(JK + KD + KPT)$</td>
</tr>
<tr>
<td>Vertical</td>
<td>No</td>
<td>$O(JK + PT_{max}\vert D^k\vert + EK(PT))$</td>
<td>$O(JK + KPT)$</td>
</tr>
<tr>
<td>Vertical</td>
<td>Yes</td>
<td>$O(JK + E(PT)(\max \vert D^k\vert + \log(K)) + EK(PT))$</td>
<td>$O(JK + PT\log(K) + KPT)$</td>
</tr>
</tbody>
</table>

5.3 Experimental Setup

5.3.1 Implementation

In order to simplify the evaluation of these algorithms, they were implemented in C++ using the MPICH [mpi07] implementation of the MPI communication standard. This approach worked well because our distributed data mining environment is simulated on a locally connected cluster. To implement the Paillier encryption scheme, the Number Theory Library (NTL) [ntl07] was used along with code adapted from [Liu07].

One practical issue that must be dealt with when using the Paillier cryptosystem is the fact that cannot naturally encrypt floating-point numbers. Floating-point numbers must be converted to a fixed-point representation. This is done by multiplying the floating-point numbers by a large constant $C$ and then truncating to an integer. In these experiments, $C = 100000$. Other methods, such as the one described in [FSW02] can also be used.

Due to the unavailability of a coordinated multi-organizational set of servers for this experiment, a cluster of servers was used for the testing. UCF’s Hilbert Cluster was used
for the final testing. The Hilbert Cluster has 64, quad-core, Intel Xeon 3GHz nodes, each with 8GB of RAM, connected to a frontend and centralized storage by gigabit Ethernet switches. This is not an unrealistic set of resources that would be utilized for this task in production. Many organizations today are using relatively inexpensive yet powerful Xeon based servers. And many organizations are connected by high performance networks, and in some cases (universities, research hospitals, etc) multi-gigabit per second dark fibers are used for communications.

5.3.2 Experimental Database

Because this paper's objective is not to assess the classification accuracy of the PNN, but to evaluate the security and efficiency of the PP-PNN algorithms, we use an artificially generated database for our experiments. An artificial database also allows us to generate data of arbitrary size, and dimensionality so as to evaluate the scale-up of the PP-PNN algorithms. The artificially generated database consists of 2-class Gaussian functions, with a 15% overlap on one of the dimensions.

5.3.3 Test Description

The performance test are intended to evaluate the scaling of the four algorithms, with separate tests for the horizontally and vertically distributed algorithms. In the horizontal tests, the number of parties is varied from 3 to 16, with each party owning 8,000 training
points. Therefore, the total number of training points varies from 24,000 to 128,000. The dimensionalities of the training points are also varied using values of 8, 16, 32, and 64 dimensions.

In the vertical tests, the number of parties is also varied from 3 to 16. The dimensionality of the training points is varied using values of 8, 16, 32, and 64 dimensions. A constant size of 128,000 training points is used to closely simulate a large scale data mining task. The dimensions of the 128,000 points are split evenly among the parties, with the last party also receiving the labels of the training points. Both the private and public query algorithms are evaluated for the time that it takes for them to execute a single test point query. A nominal $\sigma$ value of 0.5 over all dimensions and classes was chosen, because it worked well in the initial experimentation with the Gaussian dataset. Because this paper only focuses on the computational efficiency of the PP-PNN algorithms and not their classification performance, no effort was expended to optimize the sigma parameter values.

5.4 Results

In the graph for the horizontally partitioned, public query PNN (figure 5.7), scaling was quite moderate for both increased dimensionality and increased number of parties in the computation. In this graph, as in the others, some minor fluctuations can be seen, attributable to background loads on the cluster computer.
Figure 5.7: Performance of the horizontally partitioned, public query PNN. The time taken scales gradually with the number of parties, and is low in absolute terms. The scaling is moderate because there are no expensive encrypted operations, and much of the work is done in parallel by the parties. The time taken scales with the dimensionality as in the standard serial algorithm.

Clearly, of the four algorithms presented, the horizontally partitioned, public query PNN has the best performance and is best suited for fast operation in a distributed environment. This is because it avoids the use of cryptography, provides data parallelism and minimizes communication.

In figure 5.8, the performance of the horizontally partitioned, private query algorithm is shown. The time appears to scale quadratically with the number of parties and training points, likely because of number of spurious values that must be sent in order not to reveal each parties’ number of training points in each class. While the private query case remains practical to perform, there is a significant difference in performance between the private query and the public query cases, further supporting the need for differentiating on the
Figure 5.8: Performance of the horizontally partitioned, private query PNN. The algorithm scales quadratically with the number of points. Dimensionality does not affect it significantly because the time is dominated by the encrypted operations for the distance calculations, which is constant across dimensionalities in this algorithm.

privacy of the query. Code profiling of the private query case reveals that 90% of the time is spent in the large integer calculation functions needed by encrypted operations.

To further evaluate the impact of large integer calculations, we use the NTL library to perform a large number of multiplications and additions, first with standard integers, then with large, NTL integers. The results are presented in table 5.3. There is a clear difference by several orders of magnitude between the operations, causing the observed difference in performance of the two algorithms. However, because of the highly parallel nature of the PNN operations that require these encrypted calculations, and because of potential improvements to the supporting cryptosystems, these differences should be mitigated in future implementations.
Table 5.3: Average Time Integer and NTL Integer Arithmetic

<table>
<thead>
<tr>
<th></th>
<th>Standard Integer</th>
<th>NTL Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>1e-08s</td>
<td>9.79e-04s</td>
</tr>
<tr>
<td>Multiplication</td>
<td>2e-08s</td>
<td>6.57e-05s</td>
</tr>
</tbody>
</table>

In figures 5.9 and 5.10, the performance graphs are shown for the vertically partitioned public and private query, respectively. Both algorithms scale moderately with increased dimensionality, and stay mostly constant for different numbers of parties. Both are shown to be practical to use. However, the public query version again has a significant performance advantage over the private query version, for the same reasons stated for the horizontally partitioned algorithms: the private query version incurs additional encrypted operations, which are computationally intensive. Again, this reinforces the case for differentiating algorithms on the basis of query privacy.

Figure 5.9: Performance of the vertically partitioned, public query PNN. The performance remains relatively constant over the dimensionality and number of parties, because the number of demanding encrypted operations is on the order of the number of points in the training set.
Figure 5.10: Performance of the horizontally partitioned, private query PNN. The behavior is similar to the public query version, remaining relatively constant for number of parties and dimensionality, again because the number of encrypted operations is on the order of the number of training points.

5.5 Summary

In this chapter, we have shown four practical algorithms for implementing the PNN in a privacy preserving distributed environment. We have evaluated their performance on a large scale data mining task. While both PP-PNN algorithms (public-query and private-query) remain practical, clearly the public query algorithms have a performance advantage over the private query ones. To begin with, this shows that it is sensible to differentiate on the basis of query-privacy. As a rule, whenever one is allowed to expose a greater amount of knowledge, one can use that fact to potentially increase the algorithm’s efficiency. It has also become clear that where possible, one should avoid the additional expense incurred by encrypted operations.
CHAPTER 6
THE CHALLENGES OF PPDM DEVELOPMENT

Because research into PPDM algorithms is relatively young, there are still numerous challenges remaining for implementing them. Current use of PPDM algorithms, especially SMC-based ones, is limited to a narrow range of organizations and applications, most of them being experimental. Of the 25 SMC-PPDM algorithms surveyed in this dissertation, only 6 of them are presented with performance results of actual implementations, meaning most of them were likely never implemented and tested. During the development of the algorithms in chapter 5, numerous practical difficulties were encountered. To begin with, there is no standard framework to which one could turn in developing PPDM algorithms. Next, while the prototype was developed for executing on a local cluster, extending it to a wide-area, unreliable network environment presents new challenges. Finally, it is clear that from the long runtimes for some of the algorithms, that PPDM can be resource intensive. In this chapter, some of these difficulties are explored, so they can be addressed in chapter 8.

6.1 Supporting Frameworks

Despite the development of numerous SMC-based PPDM algorithms, there has been little support in terms of development models and architectures to adapt these algorithms
to practical applications. The systems available in the literature still do not demonstrate a comprehensive and easy to use framework for general development of PPDM algorithms, and do little to integrate HPC resources. TRIMF [AK06] proposes a runtime environment to support privacy preserving data access and mining, providing an ensemble of related services for PPDM. TRIMF also supports fine-grained access control where each party can specify which data is accessible and to whom. The system described in [AK03] suggests improvements to the Globus GSI security framework [FK], such as sandboxing and intrusion detection, to support PPDM. While these systems can potentially enable efficient PPDM processes, and scale to many parties, they do not suggest a clear framework with which to implement a variety of PPDM algorithms.

In [WXS05a] the authors suggest a hierarchical structure combining P2P and grid concepts in order to efficiently support PPDM. Peers within virtual organizations (VO) communicate locally, and then use super-peers to communicate among the VO. While an architecture resembling this has tremendous potential for facilitating large-scale PPDM, it is not clear exactly how a system like this would operate, and what kind of programming model it would use.

Some systems are built specifically on an automated framework of privacy preservation. Fairplay [MNP04] is a domain specific language for two-party secure computation. Fairplay generates secure circuits in a Secure Hardware Description Language (SHDL) and then executes those circuits. SMCL and SMCR [NS07], a domain-specific SMC language and runtime respectively, offer an automated way of generated secure SMC programs, without requiring the user to explicitly manage communication.
While domain specific languages have the potential to ease the development of PPDM algorithms, they also have drawbacks. The new domain specific systems present a problematic learning curve to developers who are trained in the use of standard languages. In addition, the languages discourage re-use of existing code for data mining, which can increase costs. Finally, none of the surveyed languages could take advantage of HPC resources to speed-up the PPDM process. In addition, systems that automatically generate SMC algorithms are also implicitly parallelizing the computation; automated parallelization can only find mechanical ways of distributing the computation and does not make decisions to refactor the overall computation in more efficient ways.

6.2 Computing Resource Requirements

One of the main drawbacks of SMC-PPDM methods over their data-perturbation counterparts is the high resource demands in both computation and network communication. As an example, one study [VC04b] describes building a 408-node decision tree with an SMC-PPDM algorithm from a 1728 item training set in 29 hours. Code profiling of one of the privacy preserving PNN algorithms (chapter 5) reveals that 90% of the time is spent in the large integer calculation functions needed by encrypted operations. These resource requirements highlight the need not only for more efficient algorithms, but for more capable computational architectures to support them.
To support PPDM development, there is significant work in the fields of High-Performance and Distributed Data Mining that can be of use. The next chapter discusses some of this work.
CHAPTER 7
HIGH-PERFORMANCE AND DISTRIBUTED DATA MINING

With the availability of terrabytes of data to be mined, and with more complex machine learning algorithms, parallel and high-performance data mining and machine learning has gained a significant amount of attention. Also, because of their low cost, cluster computers and grid architectures have replaced many of the proprietary and dedicated supercomputers. Therefore, there is a natural impetus for using clusters and grids for high performance data mining. In addition, the Internet creates an opportunity to merge data from geographically distributed sources to permit meaningful mining of merged data sets. This field has become known as Distributed Data Mining (DDM). Work in both of these fields has the potential to improve ease of development and performance in PPDM systems. These areas of research are discussed in this chapter, in the hope that they can provide architectures and techniques to increase the performance of privacy preserving data mining algorithms.

7.1 High Performance Parallel Data Mining on Clusters and Grids

The computation required for data mining is often significant, especially for large databases. For this reason, much research has focused on parallel and high-performance machine
learning frameworks. In this section, some systems are presented that allow machine learning algorithms to be executed in cluster, grid and Network of Workstation (NOW) environments. These environments allow organizations to execute computationally expensive algorithms on large databases, with relatively inexpensive hardware.

Systems that provide convenient abstractions for simple development within a parallel environment have received a great deal of interest in recent years. One of the most popular, MapReduce [DG04] is a simplified parallel program paradigm for large scale, data intensive parallel computing jobs. By constraining the parallel programming model to only the map function and the reduce function, the MapReduce infrastructure can greatly simplify parallel programming. The fact that a parallel machine is involved is hidden from the programmer. While popular applications have included distributed text search and analysis of server logs [DG04], it has also been adapted to support many machine learning algorithms including Locally Weighted Linear Regression, k-means, Logistic Regression, Naive Bayes, linear Support Vector Machines, Independent Component Analysis, Gaussian Discriminant Analysis, Expectation Maximization, and Backpropagation [CKL06].

A similar system, Dryad [IBY07] is also well-suited toward the data intensive processing found in many data mining algorithms. In Dryad, computations form the vertices and communication channels form the edges of a dataflow graph, which is translated into an execution plan and executed by the Dryad engine. In this way, as with MapReduce, the developer can produce data-parallel code by abstracting the parallel processing.
Stream Processing Core (SPC) [AAB06] similarly abstracts the mining of large data streams into interconnected processing elements. The developer simply implements these elements, and allows the SPC runtime to generate and execute the associated flow graphs. It joins together data streams through relational or user defined operators. The modularity of the processing elements facilitates their use across several applications.

While MapReduce, Dryad and SPC are mostly developed for a dedicated cluster environment. However, concepts within grid computing hope to make the use of computational resources, even those across organizational and administrative boundaries, as easy as drawing resources from the power grid. These systems include Networks of Workstations (NOW) architectures, which take advantage of idle machines to lower system costs. However, developing the software to support these architectures can be challenging, as they are often less reliable, less available, and have fewer resources than their dedicated cluster counterparts.

In [PS00] a system for data mining in NOWs is developed, built on a simple primitive called Distributed DOALL. Distributed DOALL can be applied to loops that have no loop carried dependencies or conflicts, loops which are frequently encountered in data mining. Workstations receive and cache data from data servers and receive tasks from the client.

DataCutter [DA02] is general purpose grid middleware for data mining problems. DataCutter relies on a programming model which represents computations as filters and communication as streams between the filters. A data parallel version of Java provides a high-level interface for DataCutter.
The system described in [JA01] emphasizes ease of development, managing large data sets, rapid evaluation of parallelization strategies and tuning of parallel performance as motivations for the middleware which they implement. The system abstracts the parallel data mining algorithms into series of local reductions that happen on each individual processor, and global reductions which occur among several processors. This abstraction is supported by a *producer-consumer* programming model.

As grid technologies become standardized, many are finding their way into data mining systems. Globus [FK] and Tomcat [tom08] are two popular software packages actively used to support data mining middleware. The system in [LBB04], describes different components for a complete grid system for Association Rule Mining, implemented in Globus. The components described include a grid tier with heterogeneous compute nodes, a service tier providing data mining scripts which are adapted to different systems, and a client/portal tier which facilitates user friendly interaction with the rest of the system.

One of the most comprehensive DDM systems currently existing is the DataMining-Grid software, which is freely available for use. It provides functionality for tasks such as data manipulation, resource brokering, and parameter sweeps. It was developed to encourage grid transparency, the adaptation of current code to be grid enabled, providing an SOA. It is also built on the Globus GT4 toolkit. The resource brokering is supported by Gridbus [BV04]. The system aims to be accessible by both experts and novices with experts able to describe a detailed workflow in Triana, and novices able to submit simple requests through a web based system. The authors emphasize that extensibility is im-
important for a system like this, with the ability to add new components without adversely affected the existing large components and implementations in the system.

Because of their dependence on a trusted computational environment, these architectures cannot be immediately adapted to a PPDM environment. However, they can still serve a support role in making PPDM processes more efficient at the intra-organization level. Those systems that can leverage the grid especially can help reduce both startup and operational costs for getting involved in collaborative data mining.

### 7.2 Distributed Data Mining

Quite often, is is impractical or undesirable to bring all of the necessary data together to one site, or to one local computing grid to be mined. Separate organizations or multiple sites of a single organization may want to collaboratively mine databases which are in different geographic locations. Distributed Data Mining (DDM) finds ways to mine databases distributed across the Internet, while working within the constraints placed by limited bandwidth and computing. DDM often involves the application of grid computing principles to manage several data and computational resources for a coordinated data mining task.

There are several different techniques and technologies used to develop DDM applications, including the use of agent-based approaches, peer-to-peer (P2P) networks, grid middleware, web services or some combination of these. We examine each in turn, discerning what advantages they can bring the DDM process.
Many DDM support systems are built on agent-based approaches. While the term “agent” is often used in conjunction with distributed systems, we will define agents systems as those either using Agent Oriented Programming (AOP) techniques or frameworks that make extensive use of mobile code in completing data mining tasks.

There are potential benefits for agent-oriented approaches to be used in large-scale DDM, including enhanced scalability because of decentralized control, the potential for integrating learning strategies into the way the agents themselves [KLM03] and simple extensibility by injecting new agents into the environment In [PC00], the authors use a voting method based on classifiers that are constructed on individual collections of the data. This is system is based on the Java Agents for Metalearning system. This is a sophisticated system which can import classifiers from other hosts, and utilize what are called bridging methods for bringing together databases that are not immediately compatible.

A system called BODHI (Beseizing knowledge through Distributed Heterogeneous Induction) [KPH99] drives a DDM process between sites with heterogeneous (and in this case, vertically partitioned) data. The system is built on the fact that any function can be composed in a distributed fashion by the appropriate basis functions. In particular, attention is give to a Decision Tree, based on a Fourier spectrum. BODHI supports the DDM process through communication facilities, independent representation of data models and code mobility.
Despite several successful agent-based systems, there are also potential drawbacks for agents include security and hijacking risks [KLM03]. In addition, the development of agents with AOP techniques is still new research territory. It is not yet clear that AOP makes development significantly easier over traditional distributed development models. Therefore, the system presented in this paper does not make use of agent technologies.

7.2.2 DDM Middleware

The rise of the popularity of grid computing and the release of the popular Globus [FK] toolkit has encouraged the creation of middleware designed to flexibly and extensibly support the development and execution of DDM algorithms.

Papyrus [BGS99] is middleware that facilitates DDM across a network of cluster computers. Papyrus can follow three strategies: Move Results, where the intermediate computations on a cluster’s data set can be moved among clusters, Move Models, where complete predictive models are moved from site to site, and Move Data, where all data is moved to a central site to be computed. Papyrus uses a cost function describing the resource and accuracy tradeoffs of each to decide on a strategy.

A system called the Knowledge Grid (K-Grid), is a comprehensive architecture and tool suite for DDM is discussed in [CCP04] and [CCT02]. The core layer of K-Grid is implemented on top of Globus services wherever possible. The core layer is responsible for managing the metadata about data sources, algorithms and mining tools, as well meeting the requirements of the data mining tools being run with the necessary grid resources.
The high level layer of the K-grid software is responsible for orchestrating the execution algorithms and the production of models. A DAS (Data Access Service) component is responsible for finding and selecting the appropriate data to be mined, while the TAAS (Tools and Algorithms Access Service) obtains the proper algorithm to be executed on the data. The RPS (Results Presentation Service) is responsible for reporting and visualization of available results. The EPMS or Execution Plan Management Service, large scale, multi-level data mining arrangements can be expressed to the system. Through the tools available in EPMS, data sources, algorithms and the resources to process them can be arranged by the user in an arbitrarily complex way. This layer also includes a tool called VEGA (Visual Environment for Grid Applications) that is a visually interactive way to compose this composition of tasks. Users can point and click on objects representing servers, data sets, and algorithms.

Discovery Net [CGG02] provides useful abstractions for both computation and data. It allows the user to specify a multi-party work flow, confederate multiple disparate data sources into single coherent ones, and match tasks to available computing resources. The authors mention that systems like this can be used to facilitate multi-party scientific workflows for discovery.

The system described in [ATS02] emphasizes scalability and portability. The system uses Java for the language, RMI for the intercommunication, XML for much of the storage, and JDBC for database connections. While this is put together into a flexible and efficient framework, the fact that a language choice is imposed may limit a company’s
ability to adapt current infrastructure to DDM environments, and therefore may hinder adoption.

DisDaMin [FOL06], a grid system to support Association Rule Mining, is built on the concept of a *Federator* node that coordinates the transfer of data and computational tasks to proxy nodes. These proxy nodes can be associated with distributed cluster computers, which then in turn coordinate the computation and finer-grained transfer of data to the cluster nodes under its control.

The authors of [nM01] advocate having the most flexible architecture possible to support DDM. They mention that new algorithms should be included easily, the system should integrate relational and data mining operators, and the system should integrate interoperability and resource management mechanisms. They argue that DDM architectures should be built not to support a specific kind of data mining paradigm (classification, ARM, clustering, etc), but should instead offer a broad base of support, much like an operating system.

With the emergence of the P2P approach in computing that avoids centralized servers in favor of decentralized peers, it follows that these technologies could be adapted to DDM. The author of [DBG06] argues that P2P networks can offer the following to DDM: scalability, availability, asynchronism, decentralization. The DHAI system [WXS05b] employs superpeers, a P2P mechanism by which more powerful parties are elected to help manage the network with the other parties. In this arrangement, supercomputers with high speed networks may serve as superpeers, while desktops and other nodes may partic-
ipate at the bottom level. However, it is not clear how easily this would be implemented and how practical it would be for development.

However, while P2P technologies add robustness to a network, they can also add a great deal of complexity. These systems can be difficult to develop for, especially with current frameworks.

### 7.2.3 Web Services for DDM

The primary use of web services in data mining is to treat access to data mining algorithms and database as a service, which can be requested on-demand. This paradigm for systems may be not only be beneficial from a technical standpoint, but from the standpoint of commercialization as well [KZL00].

A web based system for meta-learning, called WebDisC, is presented in [TBV04]. Nodes coordinated by the system are geographically distinct, with their own data and classification systems. WebDisC has as its cornerstone a portal, storing metadata about the classifiers involved in the fusion process. The example used in the paper involves assessment of a loan application by different credit agencies. The submitted loan application may be sent to all of the different nodes which will classify whether or not that application would be approved for a loan.

The system in [KKR04] uses an execution framework in conjunction with a registry of algorithms and databases to complete a large-scale data mining task, by matching tasks to be executed to available services.
In [CZW06] the authors use an auxiliary technology to web services, the business process execution language BPEL4WS, to facilitate DDM. BPEL4WS is a standard that defines both actors in a process, as well as the flow of the process in the form of a Directed Acyclic Graph (DAG). The framework afforded by BPEL4WS in terms of orchestrating execution, handling exceptions and enforcing quality of service allows the developer to concentrate more on the data mining process itself.

In [ART05] the system presented uses a Triana-based work flow editor, along with visualization tools and WEKA [WF05] algorithms, in order to compose web-services for a data mining process flow.
CHAPTER 8
APHID

Reviewing the literature and available software to support PPDM, it is clear that there is a significant barrier to developing PPDM applications. The first of these impediments is that there are no standardized libraries to support PPDM. Next, organizations would need a middleware framework to support PPDM, which is not sufficiently provided in current systems [BGS99, CCP04, SSK07]. Even with the availability of such frameworks, simple development environments are lacking; it is especially difficult to integrate the PPDM level of mining with the use of local high-performance computing resources (e.g. grid, clusters and specialized hardware). APHID (Architecture for Private and High-performance Integrated Data mining) seeks to overcome these limitations. The design is influenced by several desiderata, which have been explicitly identified in the literature or found lacking in other systems. The system must have:

1. **Low development cost** [ATS02] (by mitigating development complexity and encouraging reuse)

2. A runtime environment for executing algorithms (as in [BGS99, CCP04, SSK07, AK06, AK03])

3. The ability to leverage high-performance computing resources (such as [DG04, PS00] and others, currently not provided in any referenced PPDM runtime)
4. Flexibility to support many PPDM algorithms [nM01]

5. Support many popular languages (i.e. portability [ATS02])

6. Scaling to support numerous parties and users [ATS02]

To support low development cost and language independence, DDM/PPDM functions are provided as a collection of web services (as suggested in [CCP04, SSK07, AK06, AK03]), which can be called by the application program. To begin with, web services libraries are available for almost every popular language in use, so the services can be implemented in any language or platform, and consumed by a different language or platform. Provide a set of frequently used services reduces development effort and hence cost, for implementing a new algorithm. In charge of these services is a set of master services, the Main Execution Layer (MEL), discussed in section 8.2. MEL orchestrates the execution of the PPDM algorithm, providing the necessary runtime environment.

APHID is explicitly built on a two-tier system of PPDM, which differentiates it from other systems. On the first tier, different organizations (also called parties in the PPDM context) communicate with each other, typically using secure, privacy preserving communications. The second tier includes grids and clusters within a particular party. Treating these tiers distinctly helps the developer to manage the complexities inherent in each level (see figure 8.1 for an illustration).

The interface to the local high performance machines is also provided as a set of web services for the individual functions in the algorithm. Therefore, an algorithm can be developed once and shared among all of the parties, with the developers at each
Figure 8.1: A two tier PPDM architecture, with secure inter-party communication at the top level, and trusted, high-performance parallel and grid resources at the intra-party level. Because the data is restricted within certain parties, it is harder to make use of a fully flexible P2P architecture. However, by imposing a two-tiered layout onto the architecture, the analysis and therefore development of high-performance PPDM algorithms becomes easier.

individual party providing only what is necessary to interface with the party’s database and high performance machines. These services support the requirement of leveraging HPC resources.

Figure 8.2 shows the stack of systems comprising a typical APHID installation within a single party. Organizational data to be mined is frequently stored in a relational database server. Because a relational database manager is typically insufficient for flexible data mining, and because these servers are often intimately involved in core business processes, this data is converted and transferred to a high-performance distributed file system (e.g. HDFS [had07], or grid-based storage). This synchronization should be done periodically and at off peak times, before it is needed for a data mining process.
The PPDM process begins with a request from a client, typically as part of a larger application, for the output of a specific PPDM algorithm (e.g. a classified test point, a classifier model, a set of clusters, or a set of association rules). The algorithms are available by unique services representing the algorithm, a partitioning (vertical, horizontal, or arbitrary) and a specific implementation.

While the MEL is responsible for initiating the PPDM process, it will frequently need to use SMC-PPDM primitives (e.g. secure sum, secure scalar products) and perform compute and memory intensive operations on training data. The PPDM services layer and the High-Performance Computing (HPC) respectively support these needs. The PPDM services layer controls the top tier of the PPDM process. The HPC services layer is a generic interface that interacts with a pluggable set of cluster and grid runtime systems (e.g. MapReduce) to perform the local mining of the database which will become part of the larger PPDM algorithm. It will store and access training databases, and submit compute-intensive jobs through the appropriate channels. Having these broad collections of service-based functions available meets APHID’s requirements for flexibility.

An important goal of DDM research involves reducing costs [ATS02], in both the areas of development and resource usage. To support low development cost and language independence, DDM/PPDM functions are provided as a collection of web services, which can be called by the application program. The interface to the local high performance machines is also provided as a set of web services for the individual functions in the algorithm. Therefore, an algorithm can be developed once and shared among all of the parties, with the developers at each individual party providing only what is necessary to
interface with the party’s database and high performance machines. If the party lacks any high performance machines, then this computation can be implemented on a simple server.

Figure 8.2: A typical APHID system stack. Organizational data is transferred from an operational database (frequently a relational database server) to a cluster/grid distributed filesystem. When a client requests a PPDM algorithm service from the MEL, the MEL then directs the mining process. The MEL makes requests data intensive operations from the HPC services, which are then directed by a master server and computed by the HPC nodes. SMC operations are routed through the SMC services, which communicate with other parties.

Before describing the functions of each layer in detail, the development model around which APHID is structured is first described.

8.1 Development Model

Before focusing more closely on each layer of APHID, a development model must first be established to provide a simple yet powerful abstraction for PPDM development. Of primary importance in APHID development are both a program style similar to
many HPC frameworks, and policy-attached shared variables, which mitigate complexity and therefore development cost (item 1 of our desiderata). Research in parallel and distributed data mining has sought to ease development by introducing new simpler development models [DA02, DG04, JA01], and APHID continues in this spirit.

8.1.1 Program Structure

In order to bridge computations on a grid or cluster with DDM/PPDM computations, a simplified interface is needed. Programming hundreds or thousands of machines of a cluster, typically on local networks, along with remote DDM/PPDM sites, typically connected on the Internet, has the potential to significantly confuse a developer. To simplify development, at the cluster/grid level, parallel development environments like MapReduce are used. At the DDM/PPDM level, an Single Program Multiple Data (SPMD) style is used. SPMD is the same programming style used in implementations of the Message Passing Interface (MPI) [mpi08], which is a popular development environment for distributed programming. The SPMD style is appropriate because all parties should be able to examine the operations involved in a PPDM algorithm. For each PPDM algorithm, there should exist one copy of the code that all parties can examine, thereby ensuring security.
8.1.2 Shared Variables

One technique APHID employs to simplify PPDM application development is the use of shared variables, implemented on top of a web services framework. These variables behave similarly to those of traditional shared memory systems, with the exception that they have a particular policy attached. A policy determines how and by whom the value may be accessed.

The available policies are shown in table 8.1. The first policy Intra-Party (IP) creates an intra-party shared variable only, which is simply a handle that allows the data to be passed when needed from machine to machine in the stack. The second policy, Fully Shared (FS), creates variables for which the value can be passed and modified among some specified list of parties. Finally the Secret Shared (SS) policy creates variables for which disparate shares are split among several parties. One example of this kind of sharing is the result of a shared secure scalar product, as in [GLL04]. At the end of this operation, the two participating parties each have shares of the final scalar product value, which are each indistinguishable from random but whose sum is the full result of the operation.

The shared variables with policies afford several advantages. First, it makes the sharing and broadcasting of values among parties relatively transparent. An example is given in figure 8.3. This example is given in language neutral pseudo-code because it is implementable in any popular language. Line 1 declares a shared variable, and line 2 attaches a policy: in this case, the $V_{\text{shared}}$ is fully shared between party $P_1$ and party $P_2$. 

89
Table 8.1: Types of variable sharing policies.

<table>
<thead>
<tr>
<th>Policy</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intra-Party (IP)</td>
<td>Only shared within a party, among layers of the PPDM stack.</td>
</tr>
<tr>
<td>Fully Shared (FS)</td>
<td>Represents a variable with shared read and/or write access between at least two parties.</td>
</tr>
<tr>
<td>Secret Shared (SS)</td>
<td>Represents a variable where independent shares are given to two or more parties, which combined yield the final result.</td>
</tr>
</tbody>
</table>

In lines 3–4, executed by party $P_1$, a value of 1 is written to $V_{shared}$. When in lines 3–4, the value of $V_{shared}$ is read by $P_2$, $P_2$ automatically requests and caches that value.

```
1 float $V_{shared}$;
2 setPolicy($V_{shared}$, PolicyFS($P_1$, $P_2$));
3 if $P == P_1$ then
4   \[ V_{shared} = 1; \]
5 else if $P == P_2$ then
6   \[ R = 1 + V_{shared}; \]
7 else if $P == P_3$ then
8   \[ R = 2V_{shared}; \]
```

Figure 8.3: Example shared variable usage

Shared variables with policies offer automatic checking for authorization. Figure 8.3 also gives an example of this scenario. Suppose code is accidentally written such that it tries to access a variable locked onto another party (lines 7–8). Upon trying to retrieve this value, the runtime will throw a security exception, and the execution will typically be halted.
8.2 Main Execution Layer

The Main Execution Layer (MEL) is itself a collection of services. These are the high level services that compromise the full data mining algorithms themselves (e.g. Naïve Bayes, k-NN, ARM, etc.) which are then easily integrated into higher-level applications. The MEL also consists of the processes that are responsible for directing the execution of the DDM/PPDM algorithm.

The main PPDM execution process, executed for every PPDM algorithm request, is as follows. To begin with, a user within one of the parties (or external to the party, if they have proper access) uses the services interface to either submit a query or start the building of a model. The model can include a trained classifier, clusters, association rules, etc. Then, the party from which the query is initiated establishes a unique session ID, and registers this session ID will all of the involved parties. All of the parties then load their policies and check them against the execution. If a party’s policies are not met, it will quit the mining process and return an error. The algorithm will then call operations on its data with HPC services and perform SMC operations through its SMC services, finally yielding the requested model or query, and returning it to the user.

The MEL is supplemented by a Data Mining Execution Database (DMED), a database for information about the mining process. The DMED stores metadata about data sets to be mined and information about the execution of PPDM algorithms.
8.3 High-Performance Computing Services

Data mining systems must integrate closely with an organization’s databases, without disturbing the organization’s everyday business processes. An organization typically has one or more servers that store data, including customer purchasing records, patient records, scientific or manufacturing observations, and customer transactions, for ordinary business support. Often these data are stored in relational databases and are queried by business process applications through languages like SQL. For data mining operations, the data is often transformed and transferred to another server, which specifically supports data mining and On-Line Analytical Processing (OLAP) applications.

For interfacing with the databases to be mined, and for resource intensive computing conducted during the PPDM process, the High-Performance Computing web services (HPC-WS) provide a generic interface to this functionality. The HPC layer can be adapted by each party to interface with their specific HPC installation, which can include clusters, grids and specialized hardware.

The HPC services layer interfaces with the architecture that is responsible for large-scale database processing. The APHID installation assumes that existing of at least one data mining server, and is designed to support data processing from dedicated clusters or locally allocated grid machines. While more traditional parallel frameworks like symmetric multiprocessing (SMP) libraries or MPI [mpi08] can be wrapped in these services, using automated HPC frameworks can significantly simplify development at this layer. Systems like MapReduce [DG04], Dryad and Distributed DOALL [PS00], are well suited to simplifying the development of large-scale database operations. For the implemen-
tation of APHID discussed in this paper, a MapReduce system called Hadoop [had07] supports the database processing operations.

Within an organization’s APHID installation, the databases to be mined are periodically transferred and synchronized to the filesystem used by Hadoop, HDFS [had07] (analogous to the GFS [GGL03] in the original Google MapReduce). There is not necessarily a one to one mapping from business database to data mining sets. It may be appropriate to generate a dataset from pieces of several different business databases. It may also be more efficient to generate versions of a dataset that are in formats well suited for particular types of data mining (i.e. separate versions for clustering and frequent itemset mining). When each dataset is generated, it is cataloged through a web interface to the DMED, along with its metainformation (data provenance, intended purpose, etc).

8.4 PPDM Services

The PPDM Services (PPDM-WS) is responsible for both providing primitive SMC operations (e.g. secure sum), but also for providing the send, receive functions on which those operations are built. Developing this set of services for PPDM is efficient, because most popular SMC-based PPDM algorithms tend to utilize a small set of SMC operations. By providing a toolkit of frequently used operations, as suggested in [CKV03], developers can easily implement numerous PPDM algorithms. Table 8.2 lists algorithms which utilize popular SMC operations.
Table 8.2: Examples of supported operations at the PPDM level.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secure Sum [Sch95]</td>
<td>[YVJ06a, SGC07]</td>
</tr>
<tr>
<td>Secure Scalar Product [GLL04]</td>
<td>[JPW06, YW05, JW05, YJV06a, SGC07]</td>
</tr>
<tr>
<td>Yao Circuits [Yao86]</td>
<td>[JPW06, LP02, JW05]</td>
</tr>
<tr>
<td>Oblivious Transfer [NP99]</td>
<td>[KV03, Pin02]</td>
</tr>
<tr>
<td>Secure Matrix Products [KLR04]</td>
<td>[DHC04]</td>
</tr>
</tbody>
</table>

8.5 Smart Data Handles

Breaking the execution of a PPDM algorithm into three layers helps to modularize development, and allows the framework to scale. However, the disadvantage of this arrangement is that data must be frequently transferred between these layers. Consider, for instance, an algorithm operating on a 10 million point data set. Suppose an operation is performed which returns an encrypted distance value for each of the points. The operation would result in $10^7 \times 512 = 5\,GB$ of data. If the data must be transferred from the HPC layer, to the MEL, to the SMC layer, and finally to another party, the operation could become a significant bottleneck. Therefore, APHID mitigates the transfer of data among layers with the use of the Smart Data Handle (SDH).

A clear way of avoiding redundant transfer among layers is waiting as long as possible to transfer the result data, and then transferring it directly to the host that needs it. However, explicitly managing these transfers adds complexity to the development process, and runs contrary to APHID’s design philosophy. Therefore, the SDH is designed to abstract the details of the transfer from the developer. When APHID’s services process a computation, a reference to the data is typically returned. If the executing algorithm needs access to those results, they will be automatically returned from where they are
cached. The also SDH serves as the mechanism that supports shared variables with policies, which are the cornerstone of the APHID development model.

Each of the three APHID layers has its own cache. The HPC services cache is implemented on top of the HDFS. The caches for the other two layers are implemented in memory, although they may in the future be implemented in disk storage or a database for fault recovery and checkpointing.

8.6 Framework Implementation

APHID is implemented in Java using Apache Axis for the web service calls and Tomcat as an application server. An open source version of MapReduce called Hadoop was used [had07], which includes support for Java, C++ and Python.

Figure 8.4 illustrates the software stack of the APHID implementation. All three layers are logically distinct sets of services, and can be separated onto their own hosts. Each layer host runs a Tomcat server with Apache Axis. Whenever a PPDM application requests an algorithm service from the MEL, a new server object is created within the Tomcat container on each of the layers and associated with the request. In doing so, the executions remain distinct, and therefore more reliable. In addition, Tomcat has built in support for clustering, so the capacity of an overloaded layer can be augmented with an extra host. Each server object creates its own associated cache, used for that particular algorithm request. The HPC services layer interacts with the Hadoop job tracker server, on the same host in our implementation (although they can be separated). The data
in the DMED is currently provided by a combination of XML files and memory-based storage.

![Diagram of APHID software](image)

Figure 8.4: Software diagram of APHID. The three layers of the MEL, the HPC services and the SMC services are each contained within their own Tomcat application server. For each new PPDM algorithm requested, a new server object is made in each layer, with an associated cache.

### 8.7 Framework Evaluation

In order to show the simplicity of the APHID programming model, and to demonstrate the efficiency afforded by integrated HPC resources, an example algorithm is implemented in the framework, the public-query horizontally partitioned PNN. The horizontal PNN is well suited to take advantage of HPC resources, because of an intensive database processing step. While the presented algorithm is a classifier, it should be noted that
APHID is designed to support a number of other data mining paradigms, including clustering, outlier detection and frequent itemset mining.

The experiments evaluating these algorithms focused on two goals, the speed of the resulting system and the ease of development. While the second goal is less tangible, there should be qualitative differences in developing for such a system versus development from scratch.

The Java code given in figure 8.7 represents the single program, multiple data (SPMD) code that is executed for the horizontally partitioned, privacy-preserving PNN. The same code is executed on the MEL layers of all parties involved. The given class implements a standard PPDM algorithm interface. The MEL of each party makes a call to the calculateCCP function within the HPC services layer (through a wrapper, which can be automatically generated and can hide the details of interfacing with the web services client libraries). The implementation details of this service are given in figure 8.7. Because the main PNN program simply calls these operations as a service, each party can have a custom implementation, to interface with specialized systems (e.g. a specialized piece of data processing hardware) without modifying the main program. A handle is returned referencing the output data stored in the HPC services layer. Next, in line 7, a call is made to the PPDM services layer, to initiate a secure sum. The handle from the HPC output is passed directly to the service, so the data can be requested directly at the PPDM services layer where it is needed. An output handle from the secure sum is returned on all parties for consistency, but is only able to be read by the querying party. The code of the secure sum implementation is given in figure 8.7. The block from lines
public class PNNHPub implements PPDMAlgorithm {
    public void test(NumericalDataPoint test, TrainingSet tr) {
        Handle CCP_k = HPCWrapper.calculateCCP(test, sigmas, tr.getId());
        Handle finalCCP = PPDMWrapper.secureSum(config.findParty("P_Q"), CCP_k);
        if (config.getMyParty().getLabel().equals("P_Q")) {
            Double[] ccp = (Double[]) finalCCP.toValue(Double[].class);
            for (int i = 0; i < ccp.length; i++) {
                ccp[i] /= (Math.pow((2*Math.PI), tr.getDimensionality()/2.0) * sigmas);
            }
            int maxClass = 0;
            double maxCCP = ccp[0];
            for (int i = 1; i < ccp.length; i++) {
                if (ccp[i] > maxCCP) {
                    maxClass = i;
                    maxCCP = ccp[i];
                }
            }
            result = maxClass;
        }
    }
}

Figure 8.5: A Java implementation within the APHID framework of the main classification service for the horizontally partitioned, privacy preserving PNN. The service is executed at the MEL.

9–27 is only executed on the querying party. On line 11, the smart data handle output of the secure sum is read as an array of doubles. When the toValue is requested, the data is transferred immediately from the PPDM layer. From lines 13–26, the remainder of the algorithm is executed on the MEL.

Figure 8.7 gives the Java implementation of the secure sum service, used by the example PNN algorithm. It should be noted that the developer typically will not have to implement SMC operations, because they will mostly be able to use the ones provided already by APHID. However, the implementation is reproduced here to show that custom
SMC operations are practical to implement. The service takes as arguments the base party where the final result should end up, and a Handle wrapping an array doubles, coming from each individual party calling the function. To begin with, the function converts the $myOperands$ handle to its array of doubles (line 7). This will automatically request the data from the layer where it is housed, or from the local store if is on the same layer. Next, on lines 9–10, a new handle to be returned is declared. This declaration creates a handle with the specified label. A space is created locally within the current PPDM layer, where the data associated with this handle is stored, and from which it can be requested from other layers. The sharing type specifies that the data from the handle will only be accessible within the current stack. In the remainder of the code, lines 12–35 are executed only on the base party, and the rest is executed on the other parties. In lines 16–22, a random vector is generated and added to the operands at the base party. This is sent to the next party. At the same time, other parties are executing line 37, waiting to receive operands (with additional random elements) from their neighbors. It then adds its own operands and sends them along to the next party (line 45). This token finally reaches back to the base party, at line 26, who removes the random elements to produce the final sum. The result is stored in the PPDM layer (through the call on line 33), and becomes available to any system within the base party’s stack. It should be noted that data can be transferred both automatically through smart data handles, as well as explicitly through send and receives. These complementary operations are implemented through similar request mechanisms.
public Handle secureSum(Party baseParty, Handle myOperands) {
    Party nextParty = PPDMConfig.getInstance().nextParty();
    Party previousParty = PPDMConfig.getInstance().previousParty();
    Double[] myOperandsArray = (Double[]) myOperands.toValue(Double[].class);
    Handle toReturn = new Handle("securesum_output",
        new Policy(Policy.SharingType.INTRA_PARTY));
    if (config.getMyParty().equals(baseParty)) {
        Random random = new Random();
        Vector<Double> randVec = new Vector<Double>();
        for (int i = 0; i < myOperandsArray.length; i++) {
            randVec.add(random.nextDouble() * RANDOMLIMIT);
            myOperandsArray[i] = (myOperandsArray[i] + randVec.lastElement()) % RANDOMLIMIT;
        }
        send(nextParty, myOperandsArray);
        Double[] recvd = (Double[]) receive(previousParty, Double[].class);
        for (int i = 0; i < recvd.length && i < randVec.size(); i++) {
            myOperandsArray[i] = (myOperandsArray[i] - randVec.elementAt(i)) % RANDOMLIMIT;
        }
        toReturn.putValue(myOperandsArray);
    } else {
        Double[] recvd = (Double[]) receive(previousParty, Double[].class);
        Double[] arrayToSend = new Double[myOperandsArray.length];
        for (int i = 0; i < recvd.length && i < myOperandsArray.length; i++) {
            arrayToSend[i] = myOperandsArray[i] + recvd[i];
        }
        send(nextParty, arrayToSend);
    }
    return toReturn;
}

Figure 8.6: A Java implementation within APHID of the secure sum algorithm, executed at the PPDM services layer.
The code listed in figure 8.7 is the calculateCCP service within the HPC services layer, called by the main PNN program. The function of the service is to interface with the MapReduce master server, and to pass the necessary data to the MapReduce program responsible for the calculating the class conditional probability over the data set stored in the distributed filesystem. One challenge in interfacing with the MapReduce runtime is that the data nodes responsible for the MapReduce computation can typically only communicate with the master server and not with the rest of the servers in the APHID stack. The MapReduce program therefore requires any input data besides the training database (e.g. the test point, parameters to the algorithm) to be distributed to the compute nodes before the job begins. The libraries interfacing between APHID and the MapReduce runtime provide a helper class, MRDataPackage and its associated factory MRPackageFactory (lines 7–15). The factory takes in a number of inputs, either plain Java objects, or handles. The handles are then resolved into their associated data, and all of these inputs are serialized to the distributed file system, and made available to the instances of the MapReduce program running on the cluster. The output handle (declared lines 4–5, returned on line 21) automatically wraps the output data in a handle and deserializes it when requested. This interface provides a simple way to distribute necessary data from the APHID runtime to high-performance computing resources, while requiring minimal modifications to the MapReduce program itself. The associated MapReduce code (not shown) requires only minimal changes to deserialize the input data from the data package.
public Handle calculateCCP(NumericalDataPoint queryPoint,
        Double[] sigmas, String trainingSetId) {

    Handle returnHandle = new Handle("outputfile",
        new Policy(Policy.SharingType.INTRA_PARTY));

    MRPackageFactory packageFactory = new MRPackageFactory();

    packageFactory.startNewDataPackage();

    packageFactory.addInputObject("trainingset",
        MetadataManager.getInstance().lookupDatabase(trainingSetId));

    packageFactory.addInputObject("test.querypoint", queryPoint);

    packageFactory.addInputObject("test.sigmas", sigmas);

    packageFactory.setOutputHandle(returnHandle);

    String commandLine = makeMRCommandLine("MRPNN.jar",
        "ppdmarch.examples.MRPNN", packageFactory.save());

    MRProgramRunner(commandLine);

    return returnHandle;
}

Figure 8.7: Java implementation within APHID of MapReduce service wrapper. This service is accessible from the HPC services layer and in turn interacts with the MapReduce master server.
8.8 Performance Results

In order to evaluate the performance of the framework, an artificially generated classification test set was used. The test set is a two-class classification problem, with two Gaussian distributions separated by 15% overlap on one dimension. The results were run on a cluster with nodes containing dual 2.2GHz Opteron processors, and 3GB of RAM.

First, the performance of the APHID stack implementation is compared to the more limited MPI-based code developed in chapter 5. Both tests were performed with each party owning 8000 instances of 64 dimensional Gaussian data. The number of parties was varied from three to eight. For the APHID tests, the three layers of the stack, the Hadoop master server and a Hadoop compute node were all run within a single server node, so that in both tests one server per party was used. The MPI version of the algorithm was implemented with high-performance communication libraries, in a language (C++) that is generally more efficient than Java. However, it is reaffirming that the APHID implementation mostly keeps pace with the MPI version, closing the performance gap as the computation becomes more significant with a greater number of parties. The potential few seconds difference may be a small price to pay for significantly reduced implementation effort. And when the problem becomes larger and more computationally demanding, APHID can scale to meet those demands, as evidenced in figure 8.9.

The graph in figure 8.9 demonstrates how well the evaluated PPDM algorithms were able to take advantage of cluster resources. For these tests, the number of parties involved was fixed. Each party was given 2 million, 16 dimensional generated Gaussian data points, for a total of 6 million data points. For each party, the number of associated
Figure 8.8: Time for 3-8 parties executing horizontal public query PNN.

cluster nodes was varied from one to five. The data points were distributed to the cluster nodes prior to testing, as would be the case if the data were pulled from the production databases during off-peak times. As the graph demonstrates, increasing the number of cluster nodes can substantially decrease the amount of time for the classification operation to complete, from 87 seconds for one cluster node to 34 seconds for 5 cluster nodes. For more intensive algorithms, this difference can tremendously increase the practicality of a PPDM application.

8.9 Summary

In this chapter, APHID, a runtime and development architecture for easily implementing PPDM algorithms, was discussed. As a concrete example, the horizontally-partitioned, public query PNN was implemented in Java within the framework. The entire algorithm required relatively few lines of code (fewer than 30 in the main service, with the addi-
Figure 8.9: For 3 parties, performance results of clusters with 1 to 5 nodes for the horizontally partitioned, public query PP-PNN.

With 85 lines of a MapReduce program, and 10 for an HPC wrapper service, and remained efficient and simple to construct, despite spanning three layers for execution, several cluster nodes, and several collaborating PPDM parties. As a result of integrating high-performance computing resources, the implementation was able to process a query using 6 million training data points in well under a minute. With an architecture such as APHID in place, and a relatively low investment in implementing a new PPDM algorithm, organizations finally have a concrete support architecture to which they can turn for high-performance privacy-preserving data mining applications. It is hoped that architectures such as APHID will soon be put into practice, and open up new data mining collaborations between organizations.
CHAPTER 9
INEXPENSIVE AND EFFICIENT ARCHIVAL STORAGE FOR GRID AND DISTRIBUTED DATA MINING

When databases or other data from the mining process need to be archived, there is a potential to use grid resources to do so, both from within and outside of an organization. Doing so may be a far less costly alternative than purchasing expensive dedicated storage infrastructure. These requests can be scheduled to idle grid resources (e.g. servers, desktops, supercomputers) in order to increase efficiency and decrease cost, while maintaining some minimum availability. It should be noted that these principles of allocating storage can be more broadly applied to storage needs, for both individuals and organizations and for other kinds of data. In the journal paper where most of the work in this chapter is published ([SLB09]), this broad perspective is taken. Because of the focus of this dissertation, these algorithms are applied specifically to the challenge of archival storage for data mining.

This chapter makes extensive use of the concept of resource cost. Cost is a general term that can vary significantly with the intended application. If the archival storage allocation is for data that can be comfortably stored on third-party, remote servers, then the cost may be monetary. If the allocation is of local grid resources, the cost may represent a preference to use a certain set of resources over another (i.e. a set of idle desktops over expensive server infrastructure, which may also translate indirectly to
monetary costs). Whether or not the data can be stored on external servers naturally depends on the privacy requirements of data. For both cases, this chapter presents a set of algorithms that can compose a set of existing storage resources to meet user specified storage and availability requirements, at the lowest cost.

Note that a similar technique is used in server resources such as RAID arrays. The various RAID levels such as RAID0 (striping), RAID1 (mirroring), and so on are just different ways of combining storage resources to meet the storage, reliability, performance and cost demands. Our approach extends this model to a dynamic networked environment.

The objective is to design a broker-based architecture for the efficient allocation of the resources. As the optimal composition problem has a non-polynomial complexity, we are interested in finding efficient approximate algorithms, with modest computational requirements.

Before exploring techniques to provide resource allocations with required availability at the lowest possible cost, similar approaches in the literature are explored. First, there is work from redundant and distributed file systems, as well as from grid computing and storage in general. Finally the field of grid economics, which explores the application of economic principles to allocating grid resources, is described.
9.1 Redundant and High Performance Distributed File Systems

With the popularity of cluster computing and network computing, many storage systems have been designed to offer high-performance and reliable storage by pooling together small and less-reliable commodity components.

In [LPS05], an architecture called ClusterRAID is designed for high reliability and integrity data storage using nodes of a cluster. Redundancy information of an individual node’s data is stored in such a way as to maximize reliability and availability, while minimizing network bandwidth. In the case of a node failure, the data can be reconstructed on any spare node. The use of Reed-Solomon algorithms allow the user to specify how many faults can be tolerated. A similar system, netRAID [BP04], uses a RAID3 style storage scheme, and is able to rebuild online in the event of a node failure.

The Cluster File System (CFS) [BP05], presents a solution for distributed video storage, aiming for high reliability, low cost and transparent use. The video streams are stripped across the nodes, following the RAID paradigm. The system has the option of using additional parity blocks (analogous to RAID5) to allow the system to recover from the failure of a single node. CFS is unique in that it is optimized for the delivery of sequential video content and in the fact that the storage nodes monitor their neighbors storage node to pinpoint failure.
Traditional cooperative grids, where resources are pooled within or among organizations, need to schedule resources in ways that are both efficient and redundant.

[HRB07] presents a system for allocating fragments of databases, to provide high-bandwidth, high-parallelism and highly available access and processing. The sites are grouped by communication speed, and fragments are allocated in a way that reduces the computational cost of the potential access to the data. Redundancy of the fragments provides the increased availability. This system, however, is not intended for resources that need to be purchased from providers, and does not attempt to minimize replication through monitoring of availability.

The Athena system [JXL05] explicitly accounts for reliability in both node and links in a computational grid to tune the performance of a grid application. Athena performs a series of graph reduction algorithms to simplify the search for efficient transmission and execution paths. When a node wishes to run a program on a specific data set, Athena will use these graph reductions to calculate the most efficient path to transfer the data and the optimal nodes on which to execute the program, accounting for the reliability of the nodes and links. At runtime, it can then select the optimal path that is most likely to result in an efficient execution. While Athena was shown to be effective for execution in this environment, it does not explicitly account for pricing. It is not meant to allocate the data blocks, but merely to use the already allocated data blocks most efficiently.
9.3 Grid Economics

Buyya, Abramson and Venugopal [BAV03] discuss the shift from a system centric view of grid resource allocation, where parameters like throughput and utilization are optimized, to a view of the grid that takes the value of resources into account. They review the different economic models that have been explored in the context of grid resource allocation, including commodity market models, posted price models, bargaining, contract-nets, auctions, cooperative bartering, monopoly and oligopoly.

The models and outlets for the future grid economy are still unestablished, and it is not clear what the community will adopt. However, much of the research into economic grid resource allocation makes reference to some sort of broker ([YVB06, SAP96, PB06, BB03, BAG00, BV01, BAV03, WLL07, YSL07, YYL07] and others), suggesting that, no matter what the future grid economy looks like, brokers will play a significant role. At least in the infancy of applications that consume grid resources, allocating its own resources will place unnecessary strain and complexity on the system, which could be much more easily handled by a dedicated broker.

Because of the clear need of a broker, much research has centered around developing a broker architectures and markets. Gridbank [BB03], implements GASA (Grid Accounting and Services Architecture), which provides a variety of accounting and payment functions for resource brokering on the grid. It uses a service-oriented architecture to support grid applications, and keeps a database of producer/consumer accounts, and resource usage records. These usage records can help resource providers estimate prices for
their resources. Gridbank supports three different types of payment methods including pay before use, pay as you go, and pay after use.

In [BV01] and [BAG00], the authors present two complementary systems for economic resource allocation on the grid. The CPM (Compute Power Market) is intended to apply to low-end systems, while GRACE (Grid Architecture for Computational Economy), is intended for high end grid computing. For both systems, there are three entities, a market, a resource consumer and a resource provider. The resource providers register with the market and download a Market Resource Agents. Similarly, the resource consumers download a Market Resource Broker, that interacts with the market. The market itself mediates, potentially charging a fee.

In [YVB06] a simple directory service called Grid Market Directory is deployed to allow both application and human users to browse and query available services by type, price and many other criteria. Future applications, for instance, looking for an image rendering service would find the lowest cost service and employ that service in the application. In [WLL07] the proposed system offers different pricing schemes for general resources (such as CPU-time and storage) and specialized resources (such as access to a scientific instrument.) The system splits providers into groups, each coordinated by a separate broker. The brokers then coordinate to determine pricing for the resources. The specialized resources are priced through a double-auction.

The SX (Storage eXchange) system in [PB06] acts as a storage broker, allowing storage to be a tradeable resource. SX uses a double auction market model for open market trading and also allows storage to be exchanged. The system brokers storage requests
by taking into account capacity, upload/download rate, and time frame of the storage reservation. The authors cite many other criteria that should be taken into account, including security, high-availability, fault-tolerance, reputation, consistency and operating environment.

Even though these architectures can match up resource consumers to providers, there is still the problem that none of the providers may be reliable enough, or have the correct scale of resources needed. For instance, if a pool of desktop users are providing the resources, then it is difficult to find enough storage or CPU power for large applications. Ideally, the market could compose disparate resources (storage, CPU) into requested ones.

Because grid resources can be an unreliable and intermittent resource, there may be a need for the broker or market to compose a complex resource out a collection of simpler grid resources.

Mariposa [SAP96] aims to execute database queries across distributed servers, using an economic framework. Each client with a query specifies a budget for the query, which is then passed to a broker. The broker communicates with the various data servers, who can trade data and queries at will. When the bidding processes completes, queries are finally executed and passed back to the user.

Oceanstore [KBC00] aims to build a data storage infrastructure on untrusted servers, using both redundancy and cryptography. It is built around a market concept whereby users would pay a monthly fee for persistent and reliable storage. This would be supported by storage providers, at various locations who would trade storage resources
among themselves. The prototype implementation of Oceanstore employs both Reed-Solomon and Tornado algorithms for encoding redundancy.

While Mariposa and Oceanstore compose the necessary resources from professional providers, they are not designed to accommodate the harvesting of idle storage resources. Neither do they explicitly account for availability, which is necessary to take into account for allocating spare resources.

9.4 An Algebra of Storage Resource Composition

The main idea behind this chapter is that users requests for storage with arbitrarily high capacity and availability requirement can be satisfied through the appropriate composition of cheaper resource components which, on their own, do not meet these requirements. However, we need appropriate guarantees that a specific composition indeed meets the requirements. In addition, we need an appropriate method to find the cheapest resource composition which satisfies a certain set of requests. To achieve this, we need a way to formally describe and manipulate composed storage resources, that is, we need an *algebra of storage resource composition*. Table 9.1 summarizes the notations used in the remainder of this section.

We consider a simple or composed storage resource to be characterized by its capacity and availability. We define availability as the probability of successful resource access. We assume that a broker has control over a set of resources $R_i, i = 1...n$, each with a known capacity $C(R_i)$ and availability $A(R_i)$. 
A user requests resources from the broker by specifying the requested capacity $C(R_{req})$ and availability $A(R_{req})$. These values are determined by the user based on the requirements of his application. A user will naturally accept a resource which has higher capacity and/or availability. The broker can satisfy the resource request in four different ways. In the simplest case, the request is satisfied with a simple resource $R_{alloc}$, which has the property $A(R_{alloc}) \geq A(R_{req})$ and $C(R_{alloc}) \geq C(R_{req})$. The other three approaches are based on combining $d$ distinct storage resources through additive composition ($AC(R_1...R_k...R_d)$), redundant composition ($RC(R_1...R_k...R_d)$) or distributed error correction ($DEC(R_1...R_k...R_d)$) to produce a composed resource $R_{comp}$.

### Table 9.1: Notations for Analysis

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R$</td>
<td>Set of storage resources, each labeled $R_i$ where $1 \leq i \leq n$. Each $R_i$ storage resource includes a capacity, an availability and a price. There is only one resource per network entity even though it may be divisible.</td>
</tr>
<tr>
<td>$C(R_i)$</td>
<td>Gives the storage capacity of the resource $R$.</td>
</tr>
<tr>
<td>$A(R_i)$</td>
<td>Gives the availability probability of resource $R$, $0 \leq A(R) \leq 1$</td>
</tr>
<tr>
<td>$P(R_i)$:</td>
<td>Price per unit of storage for a storage resource.</td>
</tr>
<tr>
<td>$R_{req}$</td>
<td>A description of the resource required by the user.</td>
</tr>
<tr>
<td>$R_{comp}$</td>
<td>The resource being composed by the broker for the requesting user.</td>
</tr>
<tr>
<td>$d$</td>
<td>Number of resources chosen to be part of $R_{comp}$.</td>
</tr>
<tr>
<td>$n$</td>
<td>Number of resources accounted for in the broker.</td>
</tr>
<tr>
<td>$AC(R_1...R_k...R_d)$</td>
<td>The result of additive composition of resources $R_1$ through $R_i$.</td>
</tr>
<tr>
<td>$RC(R_1...R_k...R_d)$</td>
<td>The result of redundant composition of resources $R_1$ through $R_i$.</td>
</tr>
<tr>
<td>$DEC(R_1...R_k...R_d)$</td>
<td>The result of distributed error composition of resources $R_1$ through $R_i$.</td>
</tr>
</tbody>
</table>

Our approach took inspiration from Reliability Block Diagrams, adapting them to the concept of availability of distributed storage resources. The underlying assumption is
that failures [rel09] in individual storage components do not stop the operation of other components.

*Additive composition* combines two or more smaller resources into one larger one. All the involved resources are required to be available for the full storage resource to be considered available. The capacity and availability of an additively composed resource \( R_{\text{comp}} = AC(R_1...R_k...R_d) \) is given by the following formulas:

\[
C(R_{\text{comp}}) = \sum_{k=1}^{d} C(R_k)
\]

\[
A(R_{\text{comp}}) = \prod_{k=1}^{d} A(R_k)
\]

To additively compose a resource, the sum of the storage in the resources must be greater than or equal to our desired storage, and the product of the availabilities must be greater than or equal to the required availability.

As an example, suppose there are two available storage resources, \( R_1 \) with 0.7GB and an availability of 0.95 and \( R_2 \) with 0.3GB storage and 0.98 availability.

Therefore if both of these resources are used, \( C(R_{\text{comp}}) = 0.7GB + 0.3GB = 1.0GB \) and \( A(R_{\text{comp}}) = 0.98(0.95) = 0.931 \).

![Diagram of an additively composed resource.](image)
Redundant composition is used whenever the user requires higher availability than offered by the currently available storage resources. For this kind of redundancy (known as N-modular redundancy) only one of the resources is required to work for a user to be able to access his data. When composing a redundant storage resource, \( R_{\text{comp}} = RC(R_1...R_k...R_d) \), the capacity is constrained and the availability is given by

\[
C(R_{\text{comp}}) = \min_{k=1...d} (C(R_k))
\]

\[
A(R_{\text{comp}}) = 1 - \prod_{k=1}^{d} (1 - A(R_k))
\]

As an example, suppose there are two available storage resources, \( R_1 \) with 1.0GB and an availability of 0.8 and \( R_2 \) with 1.5GB storage and 0.85 availability. Therefore if both of these resources are used, \( C(R_{\text{comp}}) = \min(1.0GB, 1.5GB) = 1.0GB \) and \( A(R_{\text{comp}}) = 1 - ((1 - 0.8)(1 - 0.85)) = 0.97 \).

![Figure 9.2: Diagram of a redundantly composed resource.](image)

The Distributed Error Correction (DEC) is analogous to the RAID5 disk drive composition method. This model must be composed of at least three resources. All but the last resource stores some portion of the data. The last resource stores the XOR of all of the other stores. In this way, the system can lose any one resource and still provide the data to the user. For availability all or all but one of the resources must be available.
Therefore, composing a DEC resource, \( R_{\text{comp}} = DEC(R_1 \ldots R_k \ldots R_d) \), with \( d \) individual resources, requires that

\[
C(R_{\text{comp}}) = (d - 1) \min_{k=1 \ldots d} (C(R_k))
\]

\[
A(R_{\text{comp}}) = \prod_{k=1}^{d} A(R_k) + \sum_{k=1}^{d} \left( (1 - A(R_k)) \prod_{j=1}^{d, j \neq i} A(R_j) \right)
\]

As an example, suppose there are three available storage resources, \( R_1 \) with \( C(R_1) = 0.7 \text{GB} \) and \( A(R_1) = 0.8 \), \( R_2 \) with \( C(R_2) = 0.6 \text{GB} \) and \( A(R_2) = 0.85 \) and \( R_3 \) with \( C(R_3) = 0.9 \text{GB} \) and \( A(R_3) = 0.78 \).

If all three of these resources are combined into a DEC resource, \( C(R_{\text{comp}}) = 2(0.6 \text{GB}) = 1.2 \text{GB} \). For availability, \( A(R_{\text{comp}}) = 0.8(0.85)(0.78) + (1 - 0.8)(0.85)(0.78) + (1 - 0.85)(0.8)(0.78) + (1 - 0.78)(0.8)(0.85) = 0.9062 \).

Figure 9.3: Diagram of the DEC resource. In this instance, three storage resources are represented, with a reliability block diagram symbol representing the requirement that 2 out of 3 be operational.

It was previously stated that the user determines the requested resource capacity and availability from the requirements of his application. However, every application runs better with high capacity, highly available resources. To prevent applications requesting large amounts of resources with minimal benefits, the distributed storage system needs to
implement an economic model. While the details of the economic models vary, all of them establish incentives for the clients to request resources close to the actual needs of the applications and, for the broker, incentives to satisfy the requirements as inexpensively as possible.

The brokers task, to satisfy as many requests as possible for the lowest possible price is the central challenge of the system.

### 9.5 Resource allocation algorithms

The architecture and algorithms described in this section can apply to any number of cluster and grid storage/computation systems. In the APHID system, the role of the storage broker can be taken on by a system similar to the Map Reduce / GFS master server [DG04], described in the previous chapter. In a similar manner, it must monitor the available grid resources with heartbeat messages to estimate availability. Figure 9.5 illustrates how such a broker would be integrated into a local grid data mining environment.

In our notation, the broker receives a number of requests, each labeled $R_{i}^{j}_{req}$. As with the resources mentioned in section 5.1, each request has a value $A(R_{i}^{j}_{req})$ which gives the desired availability of the request and $C(R_{i}^{j}_{req})$ which gives the desired capacity of the request. The producers which are registered with the broker are in a set $R$ with the members labeled $R_{i}$ ($R_{i} \in R$). Each resource $R_{i}$, has an associated availability $A(R_{i})$, capacity $C(R_{i})$, and price $P(R_{i})$ per unit of storage. It should be noted that the needed
Figure 9.4: How storage resources are allocated. The resource scheduling server plays the role of a storage broker, matching storage requests to available resources. The data mining application requests a certain amount of storage for storing data sets or intermediate results. The scheduling server then directs the data mining application onto which grid resources it can store the data, and how much of each it is permitted to use. The scheduling server is constantly monitoring the uptime of the grid storage resources, so it can properly estimate their availability.

A storage portion can be separated from each $R_i$, leaving the rest for other allocations. The broker can meet requests by allocating portions of one or more of the resources in $R$. The broker may return an allocation that refers to portions of the storage resources from several different producers, along with the composition type of the allocation (single, AC, RC, or DEC). The resource $R_{comp}$ refers to several individual resource allocations. Individual resource allocations include a seller and an amount of the resource to be used, given by the notation $I_i = (R_i, C_{alloc}^i)$ where $R_i$ is the seller’s resource referred to and
\( C_{alloc}^i \) is the amount of that resource allocated. As with the single resources, composed resources have associated availability \( A(R_{comp}) \), capacity \( C(R_{comp}) \), and price \( P(R_{comp}) \).

In the following we describe the algorithm used by the broker to compose and allocate resources. This brokering algorithm is composed of four other algorithms, designed to find a particular kind of allocation (standard, AC, RC or DEC). The main brokering algorithm applies these four algorithms to the current request and the current collection of producers and chooses the least expensive resource that meets the requirements of the request (assuming a solution exists and can be found by the algorithms).

Let us start by introducing some notations used in the pseudocode:

- \( \text{SingleRes, RCRes, ACRes, DECRRes} \): the total resource allocations returned by the single resource search, the redundant composition resource search, the additive composition resource search, and the distributed error composition resource search, respectively. Each total resource allocation can consist of a number of individual resource allocations.

- \( I_i = (R_i, C_{alloc}^i) \): an individual resource allocation, represented the amount of resource allocated \( C_{alloc}^i \) from seller \( R_i \).

- \( P_{\text{single}}, P_{AC}, P_{RC}, P_{DEC} \): the calculated prices of the best resource allocations returned by the single resource search, the additively composted resource search, the redundantly composed resource search and the distributed error correction resource search, respectively.
• **LowestCostResource**: the composed resource allocation that is found to be the lowest cost.

• $P_{\text{lowest}}$: the calculated price of the lowest cost full resource allocation $\text{LowestCostResource}$.

Next, we list the set of subroutines used by the broker’s algorithm.

• **FindLowestSingle**: Performs a linear search through the list of producers to find the seller with the lowest price that still meets the minimum requirements.

• **FindLowestRC**: Transforms the problem of finding the least expensive RC composition into the 0/1 knapsack problem, and solves it using a well known dynamic programming approach.

• **FindLowestAC**: Uses a heuristic to find the lowest cost resource made by AC.

• **FindLowestDEC**: Uses a genetic algorithm to find the lowest cost resource made by DEC.

• **SortByDifficulty**: Sorts the list of currently queued requests by the difficulty criterion, that is $C(R_{\text{req}}^i)/(1 - A(R_{\text{req}}^i))$ with easiest requests first.

The pseudocode for the general broker’s algorithm is described in Figure 9.5. To begin with, the algorithm sorts all of its current requests, by the difficulty criterion $C(R_{\text{req}}^i)/(1 - A(R_{\text{req}}^i))$. This criterion is directly proportional to the requested capacity (larger requests have a larger difficulty criterion) and inversely proportional to $(1 - A(R_{\text{req}}^i))$ (higher availability requests will also have a larger difficulty criterion). Dealing with difficult requests first ensures that they have the first pick of what is available, making it more
likely that they can be met. Easier requests can then be met (even if at higher expense). Therefore, the broker can potentially meet more of its requests.

The general brokering algorithm searches through the available resources, as well as the possible composed resources, for resources that optimally meet the user’s requirements. It applies four separate algorithms which are each designed to find allocations of different types. The algorithm then finds the least expensive of all the potential allocations (argmin in the pseudocode). We now present each of the search functions in turn.

### 9.5.1 Redundant Composition Algorithm

The first algorithm is designed to find whether there are any RC compositions possible to meet the current request among the available resources and if so, which one is the least expensive. First, we show that the problem of finding the optimally priced redundantly composed storage resources can be reduced to the classical knapsack problem. The problem is cast as follows. There are several items $i$, numbered from 1 to $n$ to be placed
in a knapsack of capacity \( c \). Each item has an associated price, \( p_i \) and an associated weight, \( w_i \). The goal is to maximize the value of the knapsack contents while staying within the capacity. If we assume that an item must be taken or not taken, the problem is then referred to as the 0-1 Knapsack problem.

Redundant composition can be reduced to the knapsack problem. The items map to the storage resources, \( R_i \), and the price \( p_i \) is instead \( P(R_i) \). The capacity \( c \) and the weights \( w_i \) are represented by expressions of the availability.

First, all producers that have \( C(R_i) \geq C(R_{req}) \) are selected. Because the full data must be replicated when it is stored with RC, it is clear that any resource without the full requested capacity \( C(R_{req}) \) can be automatically excluded. Using the equation of availability for redundant composition:

\[
1 - \prod_{k=1}^{d} (1 - A(R_k)) \geq A(R_{req})
\]

easily becomes

\[
\prod_{k=1}^{d} (1 - A(R_k)) \leq (1 - A(R_{req}))
\]

For reasons that will become clear later, both sides must be greater than 1. A good way to guarantee this is by multiplying by

\[
\frac{1}{(1 - \max (A(R_k)))^d}
\]
This yields:

\[
\prod_{k=1}^{d} \frac{1 - A(R_k)}{(1 - A(R_{max}))^d} \leq \frac{1 - A(R_{req})}{(1 - A(R_{max}))^d}
\]

Now, this equation is turned into a linear weight function, by taking the natural logarithm.

\[
\sum_{k=1}^{d} \ln \left( \frac{1 - A(R_k)}{(1 - A(R_{max}))^d} \right) \leq \ln \left( \frac{1 - A(R_{req})}{(1 - A(R_{max}))^d} \right)
\]

If the term \( x_i \) is added either equal to 0 or 1, depending on whether the resource is included, and added over all \( n \) instead of \( d \), the problem is now the 0/1 knapsack problem, with a weight function \( w_i = \ln \left( \frac{1 - A(R_k)}{(1 - A(R_{max}))^d} \right) \) and \( c = \ln \left( \frac{1 - A(R_{req})}{(1 - A(R_{max}))^d} \right) \).

\[
\sum_{i=1}^{n} x_i \ln \left( \frac{1 - A(R_i)}{(1 - A(R_{max}))^d} \right) \leq \ln \left( \frac{1 - A(R_{req})}{(1 - A(R_{max}))^d} \right)
\]

While the aim for the original knapsack problem is to maximize the price of the included items, our aim is to minimize price. We can easily adapt the knapsack algorithm used to minimize price instead.

The general knapsack problem is NP-hard and the decision version is NP-complete [KPP04]. However, for integer knapsack capacity, a pseudo-polynomial approach does exist that uses dynamic programming [Bel57, GN72, Hu69]. To find an optimal packing takes time \( O(nc) \) [KPP04]. If the weight functions \( w_i \) and \( c \) are mapped from their real values into integer weights, then the dynamic-programming-based knapsack algorithm
can be used to approximately find the optimal RC allocation. However, for the equations to become correct, we must try sufficient numbers of \( d \) from 2 to \( n \) in the expressions
\[
w_i = \ln \left( \frac{1 - A(R_k)}{1 - A(R_{max})} \right)^d\]
and
\[
c = \ln \left( \frac{1 - A(R_{req})}{1 - A(R_{max})} \right)^d.\]
Therefore, the final RC allocation algorithm iterates over values of \( d \) from 2 to \( n \) and then applies the dynamic-programming-based knapsack algorithm to the seller resources. This algorithm allows the problem to be solvable in pseudo-polynomial time (dependent not only on the number of resources \( n \), but on the value \( c \))
\[
O \left( n^2 \log \frac{1 - A(R_{req})}{1 - A(R_{max})} n \right).
\]

The dynamic programming algorithm for solving the knapsack problem works as follows. Begin with a table of weights \( W[1..n, 0..c] \) where \( n \) is the total number of producers who are eligible to be included (have enough capacity) and \( c \) is the integer representation of the availability quantity \( \ln \left( \frac{1 - A(R_{req})}{1 - A(R_{max})} \right)^d. \) Every value \( W[i, j] \) in the table will contain the maximum value that can be included if \( c = j \). Looking in this table will reveal the appropriate resources to be included. For more information on this classical solution to the knapsack problem see [KPP04, Bel57, GN72, Hu69].

### 9.5.2 Additive Composition Algorithm

To find the optimal additively composed resource, a second constraint is added to the knapsack problem. Suppose that, in addition to not exceeding the capacity of the sack, one must not exceed another arbitrary dimension either (e.g. length). This is referred to as the 2 dimensional knapsack problem, which can also be extended to an arbitrary dimensionality, becoming the \( d \)-dimensional knapsack problem.
For additive composition, the resources that are selected when put together, must meet or exceed $C(R_{req})$. The composed resource also must not exceed the availability constraints. As stated before, the availability that must be met is

$$\prod_{k=1}^{d} A(R_k) \geq A(R_{req})$$

By taking the reciprocal and the natural logarithm, we yield an arrangement that is compatible with the knapsack weight equation, as before:

$$\sum_{k=1}^{d} \ln \left( \frac{1}{A(R_k)} \right) \leq \ln \left( \frac{1}{A(R_{req})} \right)$$

However, we must add the additional constraint that

$$\sum_{i=1}^{d} C(R_i) \geq C(R_{req})$$

This yields a 2-dimensional knapsack problem, which is not easily solvable. Therefore, we design a heuristic that takes into account knowledge of the domain.

To begin developing a heuristic, we first cull out resources that cannot be part of this kind of allocation. It is known that all producers with $A(R_i) < A(R_{req})$ can be automatically excluded, because they would immediately cause the availability of the allocation to drop below $A(R_{req})$. The AC algorithm then begins iterating through the remaining resources, starting with ones that are heuristically determined to be more
promising. We now devise a metric that gives preference to resources that would be better suited for AC allocations, called the AC criterion.

The AC criterion is defined to be:

$$ \sigma(R_i, R_{req}^j) = \frac{A(R_i)}{P(R_i)} \cdot \left( \frac{\min(C(R_i), C(R_{req}^j))}{C(R_{req})} \right) $$

The first part of the product $\frac{A(R_i)}{P(R_i)}$ gives the availability per unit price. This term is included give more emphasis to resources that have good availability relative to their cost. Having a somewhat higher availability is important in AC allocations, because availability of the composed resource drops as the product of its constituents ($\prod_{k=1}^{d} A(R_k) \geq A(R_{req})$).

The second part of the term $\left( \frac{\min(C(R_i), C(R_{req}^j))}{C(R_{req})} \right)$ becomes a factor from $[0, 1]$ giving more emphasis to producers with large allocations available. However, the $\min$ function limits this influence to $C(R_{req})$, because there is no advantage to having more space than is required when finding an AC allocation. This term helps find fewer large resources, instead of many smaller resources to again avoid a rapid reduction in availability.

- **FindAllGreaterOrEqual**: returns a list of producers with availability greater than or equal to the specified availability.
- **SortByStripingCriterion**: sorts the list descending by the calculated $\sigma$.
- **currentAllocation**: A composed resource made of a set of individual resource allocations.

The algorithm takes a set of seller resources $R$ and the requested resource $R_{req}^j$. As stated before, producers without the minimum availability are first culled out of the
Procedure: FindLowestAC

1. \((R, R_{req})\)
2. \(sellersWithMinA = \text{FindAllGreaterOrEqual}(R, A(R_{req}^j))\);
3. \(\text{SortByStripingCriterion}(sellersWithMinA, C(R_{req}^j))\);
4. real \(\text{accumulatedSpace} = 0\);
5. real \(\text{accumulatedAvailability} = 1\);
6. boolean \(\text{done} = \text{false}\);
7. boolean \(\text{valid} = \text{true}\);
8. \(\text{currentAllocation} = \emptyset\);
9. while \(\exists R_i \in sellersWithMinA \text{ and } \text{done} \neq \text{true} \) do
10. if \(\text{accumulatedSpace} + C(R_i) \geq C(R_{req}^j)\) then
11. \(\text{done} = \text{true}\);
12. end
13. if \(\text{accumulatedAvailability} \times A(R_i) < A(R_{req}^j)\) then
14. \(\text{done} = \text{true}\);
15. \(\text{valid} = \text{false}\);
16. end
17. if \(\text{valid}\) then
18. \(\text{currentAllocation} = \text{currentAllocation} \cup I\);
19. \(\text{accumulatedSpace} += C(R_i)\);
20. \(\text{accumulatedAvailability} *= A(R_i)\);
21. end
22. end
23. return \(\text{currentAllocation}\);

Figure 9.6: Heuristic algorithm for finding an additively composed set of resources.
group because they would immediately bring the availability of allocation below the requested level. Then the algorithm sorts the remaining resources in descending order by the defined AC criteria. Following the sort, the algorithm iterates through and includes available resources until its has found a satisfactory composed resource, or until there are no more resources with the minimum availability remaining. For each resource, the algorithm checks to see if its addition would help the resource meet the necessary requirements. If a valid resource is found, the algorithm then stops searching and returns the resource. Otherwise, it will return a null resource.

The loop through each of the $R_i$ is an $O(n)$ process, while the sort is an $O(n \log n)$ process, making this algorithm work in $O(n \log n)$.

9.5.3 Distributed Error Composition Algorithm

Finding the optimal distributed error composition configuration is a complex integer programming problem. However, there are a number of meta-heuristics to which we can turn. Tabu search and genetic algorithms have both been successfully applied to knapsack problems [CB98]. While our problem is harder than the traditional knapsack problem, it is relatively easy to frame in terms of a genetic algorithm. The chromosome is formed with a binary gene for each available seller. The GA then tries to maximize a fitness function which has been engineered to reward solutions meeting the minimum size, whose producers have the minimum required capacity and solutions that meet the requested availability. The fitness function is defined as follows:
Fitness(DEC) = \frac{D_1}{A(R_{req})} \min(A(DEC), A(R_{req})) + D_2 \left( 1 - \frac{d_{suff}}{d} \right) + D_3(ProperNumber(d)) + \frac{D_4}{P}

where \(D_1, D_2, D_3\) and \(D_4\) are positive constants, \(P\) is the price per unit of the entire allocation, \(d\) is the number of total producers in the allocation and \(d_{suff}\) is the number of resources with sufficient space to be part of the allocation. \(ProperNumber\) rewards solutions that are within a probable range of valid DEC solutions and is defined as follows:

\[
ProperNumber(d) = \begin{cases} 
1 & 3 \leq d \leq DEC\_MAX \\
0 & \text{otherwise}
\end{cases}
\]

\(DEC\_MAX\) (in our case 10) is a parameter of the broker representing the practical limit of the number of different \(R_i\) resources that can be part of \(R_{req}\). The number of storage resources \(d\) that are part of \(R_{req}\) could conceivably be as large as \(n\), but \(DEC\_MAX\) is used to find practical solutions and computationally simplify the task. Because this GA can produce individuals that are not valid solutions, they are checked for validity before being added to the list of possible solutions.

The fitness function of the genetic algorithm used for DEC must find solutions that are not only valid (fulfill the requested availability \(A(R_{req})\)) but minimize the price. Early experiments which only selected on the criteria of price had trouble finding valid solutions. Therefore, additional terms are added to the fitness function which reward solutions for being closer to correct. In addition to the fourth term, which selects for price, terms one to three help bias the search toward valid solutions.
The first term of the fitness function, \( \frac{D_1}{A(R_{req})} \min(A(DEC), A(R_{req})) \), evaluates how close the solution comes to meeting the requested availability \( A(R_{req}) \). By taking the minimum of \( A(DEC) \) and \( A(R_{req}) \), it ensures that the value remains between \([0, 1]\). From the standpoint of the broker, there is no need to obtain a solution with greater than requested availability. The constant \( D_1 \) provides a bias for this criterion.

The second term is designed to find solutions where all of the included seller resources have enough capacity to support the requested resource. Allocations where the included resources do not all have enough capacity are technically invalid, but rewarding closer solutions helps the GA concentrate its search. Dividing the number of sufficient-capacity resources over the total included resources, determines what percentage of the resources have sufficient capacity.

The third term is designed to ensure that the solution contains a reasonable number of different producers. Because redundancy in a DEC allocation is entirely contained within one extra allocation, availability will quickly drop for allocations of too many producers. The term ensures that the number of producers in the allocation meets the minimum number of producers for DEC (3) and is less than or equal to the maximum number of producers (\( DEC_{\text{MAX}} \)). While solutions that are not within this range are technically invalid, giving them no fitness discourages the genetic algorithm from pursuing solutions that are close to correct.

The fourth and final term helps the genetic algorithm search for smaller prices. Finding the lowest price (assuming that a valid solution has been found) is the main objective.
Therefore, smaller prices will drive the fourth term to be larger, with appropriate bias by $D_4$.

9.6 Experimental study

9.6.1 Experimental setup

In the following we describe an experimental study which measures the benefits of our proposed approach compared to a standard approach which involves only simple resources. We assume that the broker queues up a batch of requests and clears them in a single allocation step. This way more efficient allocations can be made compared to the case when a single requests is allocated at a time. In our experiments we assumed that a batch of 50 requests are cleared in an environment with 200 producers.

We compared two approaches:

- Standard allocation: the broker performs a linear search on the single resources to meet the requests.

- Improved allocation: the broker performs a search on the single resources and the improved search algorithms described in Section 9.5 for the three types of composed resources.

The data sets considered in our experiments were taken from the work of Anderson and Fedak [AF06] concerning the statistical properties of the hosts participating in the
SETI@Home project. We generated the list of storage resources available was generated by randomly selecting 50,000 free disk space \(d_{\text{free}}\) values from the host database. We filtered out values above \(10^{14}\) bytes, as these are probably due to erroneous reporting. Unfortunately, there is no host availability data provided on per-host basis by the publicly available host database. Therefore, we generated some artificial values which match the statistical properties of the SETI@Home hosts. We considered three reported values: \(on\_fraction\), the fraction of time that the SETI@Home client runs, the \(connected\_fraction\), the amount of time that the client program has access to an Internet connection and finally, the \(active\_fraction\) shows the amount of time that the client is allowed to run. For our application, we consider the average availability to be the product of these three variables:

\[
\bar{A} = (on\_fraction)(connected\_fraction)(active\_fraction)
\]

\[
= (0.81)(0.83)(0.84) = 0.5647
\]

Using the value as the mean of a Gaussian random distribution with a standard deviation of 0.1, availability values were generated and paired with randomly selected disk space values. Finally, prices per Gigabyte of space were generated for each resource pair \(R_i\). We assume that the pricing per unit of storage space is dictated by the availability, which becomes asymptotically more expensive as availability approaches one:

\[
P \sim \frac{1}{1 - A(R_i)} \quad (9.1)
\]
This pricing function is reasonable for allocation both of internal grid resources and externally provided third party resources. Within an organization, the servers with the highest uptime typically contain expensive equipment and are difficult to maintain. By the same token, external storage services with extremely high availability are also typically more expensive to maintain.

To provide random variations in pricing, this calculated price is multiplied by a normally generated random price factor with average of 1.0 and standard deviation of 0.1.

For each clearing iteration within a run with particular values for the two parameters, 200 producers were randomly selected from the loaded resource pool, and 50 consumers were randomly generated. For each set of parameters, this clearing iteration was performed 10 times and averaged. The consumers were randomly generated with average requested availabilities $\overline{A}(R_{req})$ in the range of 0.3 to 0.99 and average requested capacity $\overline{C}(R_{req})$ of 1GB, 10GB, 100GB and 300GB. With these parameters, consumer requests were generated by Gaussian random generators, with the given averages and standard deviations of $\frac{A(R_{req})}{8}$ and $\frac{C(R_{req})}{2}$ for the required availability and capacity respectively. These were chosen under the assumption that the availability required by users would not vary as widely as the space required (most users want “pretty good” availability).

The GA responsible for finding Distributed Error Compositions run for a 100 generations with a population size of 100. The default JGAP mutation rate of 0.1 was used. The constants for the fitness function were, $D_1 = 25.0, D_2 = 25.0, D_3 = 200.0, D_4 = 50.0$. 

134
Figure 9.7: Percentage of successful allocations, using the standard and the improved allocation algorithm, for various average request sizes.

9.6.2 Results

Figure 9.7 shows the percentage of allocation successes for four different average request sizes. We plot both the standard and the improved algorithm. We expect the improved algorithm to perform at least as well as the standard algorithm, as the improved algorithm subsumes the previous one. The question is whether the improvement in the allocation success justifies the considerable computational expense of the improved algorithm.
The first observation, applicable to both the standard and the improved algorithm, is that the success rate becomes lower with the increase on the average requested availability $\overline{A}(R_{req})$, as the requests become increasingly hard to meet. Second, the higher the average requested capacity $\overline{C}(R_{req})$, the lower the success rate.

For cases when the both average requested availability and requested capacity is low, both algorithms can guarantee close to 100% success rate. For other cases, however, the improved algorithm considerably outperforms the standard algorithm. In fact, for a considerable range of scenarios, the improved algorithm can maintain success rates close to 100% even when the success rate of the standard algorithm is very low. For instance, for requests with average capacity of 10GB and availability 0.9, the success rate of the improved algorithms is close to 100%, while for the standard algorithm is around 20%.

Still, the success rate of the improved algorithm degrades for scenarios which have both high requested capability (100GB and above) and high requested availability (0.7 and above). Even in these cases, the improved algorithm outperforms the standard algorithm with a significant margin, which justifies the additional computational resources.

Figure 9.8 presents the price of the allocated resources function of the requested average availability and for various values of the average requested size. Both algorithms try to minimize the cost of the allocated resources. As the improved algorithm subsumes the standard one, the average prices for the improved algorithms will be at least as low as that for the standard one.
Figure 9.8: Prices for allocations with single and multiple resources.
Figure 9.9: Percentage of compositions for runs at different average request sizes.
The overall space of the graphs reflects, in broad lines, the pricing model of Formula (9.1). Nevertheless, the average price for satisfying the customer requests can be higher or lower than the one given by the price function under the following conditions:

- The price will be higher if there are not enough resources to satisfy the requests at the desired availability level and the broker needs to satisfy it with higher availability, more expensive resources.

- The price may be lower if the broker is able to satisfy the request using redundant or DEC composition. This is true only for certain types of size, availability and composition type combinations.

These considerations show that the improved algorithm should be able to guarantee lower prices. Indeed, the graphs in Figure 9.8 validate this conjecture. However, the difference between the methods is significant only in the cases when the requests have a high average availability and are of a medium 10-100GB size, where the savings can be as high as 50%.

Next, let us study the relative contribution of the different resource allocation types to the satisfaction of the requests. Our objective, is again, to study whether the improved algorithm is justified. If a very large percentage of requests are satisfied using single resource allocations, then the additional computational complexity of the improved method is not justified. The graphs in Figure 9.9, however, show that this is not the case. In particular for the “difficult” allocation cases (with high availability and large capacity requests) most of the requests are fulfilled with composed resources (mostly DEC and
Redundant). For small average requested sizes, standard allocations at low availability can be cheaper, and both DEC and standard allocations require less redundancy (and therefore, potentially less cost) than redundant composition. This makes them dominant at low required availabilities. For the larger requested sizes (in Figure 9.9c and 9.9d), the standard allocations are not as dominant, even at low required availabilities. This is because as the average requested size becomes larger, large enough single resources are less likely to be found, and DEC has the most efficient redundancy. As availability requirements are more stringent, anything but redundant composition has less and less chance of meeting it.

We note that only a very small percentage of the requests were fulfilled using additive composition. This is due to the fact that the availability of an additively composed resource declines rapidly, and the average availability of the seller host pool is already somewhat low. It is also more rare, especially at small sizes, to find less expensive additive compositions because it takes several higher availability resource to make one lower availability resource.
CHAPTER 10
CONCLUSION

This dissertation has discussed the motivation, design, implementation and improvement of an architecture to support the development and execution of privacy preserving data mining algorithms. The background of PPDM, as well as an associated design process were discussed. A suite of algorithms for the privacy-preserving versions of the Probabilistic Neural Network were implemented, which will allow the powerful neural network architecture to be employed in multi-organizational classification applications (e.g. disease prediction with combined medical data sets). The implementation of this PNN algorithm provided insight about the difficulties inherent in the PPDM development process and those lessons motivated the development of the APHID architecture. The APHID architecture was then presented, which offered a collection of organized services to support PPDM development, as well as mechanisms to simplify and increase the efficiency of implementation. From implementation and testing of two PNN algorithms within the new framework, it was clear that the architecture was useful in employing cluster computing resources to speed up the PPDM process. Finally, different data scheduling algorithms were explored that will help make the cluster and grids on which APHID will run more efficient.

PPDM algorithms have the potential to dramatically encourage the production of vital knowledge for both individuals and organizations. It is not simply that the addition
of privacy preserving techniques will make existing data mining applications less of a concern to governments and individuals. Privacy preserving technologies have the potential to fundamentally change the types of applications that we will consider as possible. For instance, if users were confident in the ability of web-based systems to preserve the privacy of contributed data, they may be willing to share information that they would never dream of sharing otherwise (e.g. private medical conditions, salaries, etc.)

There are still many remaining challenges in the practical implementation of PPDM algorithms. Security must be assiduously maintained, because once privacy is compromised, it may be impossible to get back. Furthermore, there is much work to be done in improving the practical mechanisms (e.g. SMC techniques, cryptosystems, and available development libraries) to increase the reliability and drive down the cost of PPDM applications. Oliveira and Zaïane [OZ04] contend that there are three landmarks that characterize the field of PPDM: the Conceptive landmark, where the conflicts between privacy and knowledge discovery are explored, the Deployment landmark, the current period where PPDM techniques are published in research venues, and the Prospective landmark, where there are efforts to establish standard techniques and frameworks to support PPDM. As we enter this third landmark of PPDM, it is important for us to emphasize a practical, engineering perspective for these systems. It is hoped that the work in this dissertation has encouraged this transition from theoretical study, to practical implementation.
CHAPTER 11
FUTURE WORK

This work in developing systems to support DDM and PPDM is by no means complete. There is much work to be ensure that PPDM applications become practical. There are many direct improvements to the APHID system that can be made, as well as further study into increasing the efficiency and availability of the cluster and grid computing resources that support APHID.

11.1 Enhancements to APHID

The availability of a library of frequently encountered operations could greatly accelerate development. Future APHID implementations should also provide services for data perturbation based PPDM algorithms by supplementing the SMC layer.

Communication efficiency is also a significant challenge with web services architectures. While these communication paradigms allow unprecedented interoperability, they can perform slowly [CGB02]. Many efforts are currently underway to support data standards that are both interoperable and efficient.

Another challenge to be solved in APHID is a way of seamlessly integrating multiple sites that belong to one party. It is common for an organization to have several satellite
offices, which may cross continents. Because satellite offices within an organization can typically share data, there should be a third tier of data mining to allow this to happen more efficiently, without adding significant complexity to the programming model.

Because of the rapidly expanding popularity of cluster and grid computing, it is likely that future software will make it even easier to develop programs for data mining processing. Languages like Sawzall [PDG05] and Pig [pig08] offer high level interfaces to querying and processing data, which further abstract the process of parallel development, making it more accessible for developers. Such advances can easily be integrated into the developed architecture.

The ultimate effectiveness of frameworks like APHID will be constrained by the utility of the SMC operations that it supports. Therefore, advancement in PPDM will depend significantly on the creation of new, more advanced SMC techniques. These will include more flexible and more efficient cryptosystems.

11.2 More Efficient Usage of Grid Resources

Future work should also focus on improving the performance of the resource scheduling algorithms in chapter 9. They should include more complete understanding of the resources (e.g. network bandwidth, as in [YSL07]) as well as a greater understanding of the communication and computation patterns of the algorithms they support. More sophisticated approaches employing more fine-grained views of availability (for instance,
recognizing that resources are far more likely to be busy during the day) as well as predic-
tive models of resource behavior will further improve system reliability and availability.
Here we summarize relevant notation from the paper. Table A.1 contains notations explained in chapter 3. Table A.2 contain notations explained in section 5.1. Table A.3 contains notations explained in section 5.2.1. Table A.4 contains notations explained in sections 5.2.3 and 5.2.4. Table A.5 contains notations used in sections 5.2.5 and 5.2.6. Table A.6 contains used throughout chapter 5.

### Table A.1: Notations for Analysis, chapter 3

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v^k$</td>
<td>In secure sums and tree structured operations, the value residing at $P^k$.</td>
</tr>
<tr>
<td>$sk$</td>
<td>A private (secret) cryptographic key.</td>
</tr>
<tr>
<td>$pk$</td>
<td>A public cryptographic key.</td>
</tr>
<tr>
<td>$cx$</td>
<td>Vector of encrypted text.</td>
</tr>
</tbody>
</table>

### Table A.2: Notations for Analysis, Section 5.1

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D$</td>
<td>The dimensionality of data points in the classification problem, agreed upon by all parties.</td>
</tr>
<tr>
<td>$PT$</td>
<td>The number of points in the training set $S$.</td>
</tr>
<tr>
<td>$c_j$</td>
<td>The $j$th class of the classification problem of interest; $j = 1...J$.</td>
</tr>
<tr>
<td>$PT_j$</td>
<td>The number of points in the training set belonging to class $c_j$.</td>
</tr>
<tr>
<td>$J$</td>
<td>The number of classes in the classification problem.</td>
</tr>
<tr>
<td>$X_r$</td>
<td>The $r$th training point from $S$ that is of class $c_j$.</td>
</tr>
<tr>
<td>$x$</td>
<td>A $D$ dimensional test point in the classification problem of interest.</td>
</tr>
<tr>
<td>$x(i)$</td>
<td>The $i$th component of the training point $x$; $i = 1...D$.</td>
</tr>
<tr>
<td>$p(c_j</td>
<td>x)$</td>
</tr>
<tr>
<td>$p(x</td>
<td>c_j) = CCP_j$</td>
</tr>
<tr>
<td>$p(c_j)$</td>
<td>The a-priori probability that a datum in the classification problem is of class $c_j$.</td>
</tr>
<tr>
<td>$\sigma = (\sigma_1, \sigma_2, ..., \sigma_D)$</td>
<td>A vector of parameter values used by the PNN algorithm to estimate the class conditional probabilities for the classification problem of interest.</td>
</tr>
</tbody>
</table>
**Table A.3: Notations for Analysis, Section 5.2.1**

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>The set of training data for the classification problem.</td>
</tr>
<tr>
<td>$k$</td>
<td>A value set at each particular party with its ID. This helps it to execute the proper branch in an SPMD program.</td>
</tr>
<tr>
<td>$P^k$</td>
<td>The $k$th party of the distributed environment.</td>
</tr>
<tr>
<td>$K$</td>
<td>The number of parties in the distributed environment.</td>
</tr>
<tr>
<td>$P^q$</td>
<td>The party that generates the query requesting the prediction of the class of test point $x$; $q$ can be any one of the indices from the set ${1, 2, ..., K}$</td>
</tr>
<tr>
<td>$P^c$</td>
<td>The party that holds the classification $C(X_r)$ for every training point $X_r$ in the vertically partitioned cases.</td>
</tr>
<tr>
<td>$S^k$</td>
<td>The portion of the training set $S$ owned by party $P^k$ in the distributed environment, $k = 1...K$</td>
</tr>
<tr>
<td>$PT^k_j$</td>
<td>The number of points in training sub-set set $S^k$ of class $c_j$.</td>
</tr>
<tr>
<td>$X^k_j_r$</td>
<td>The $r$th training point from the set $S^k$ that is part of class $c_j$.</td>
</tr>
<tr>
<td>$X^k_i$</td>
<td>The $i$th component of the training point $X^k_i$; $i = 1...D$</td>
</tr>
<tr>
<td>$C(x)$</td>
<td>The class that PNN algorithm predicts for the test point $x$.</td>
</tr>
<tr>
<td>$D^k$</td>
<td>The set of dimensions owned by $P^k$ in the vertically partitioned problem.</td>
</tr>
</tbody>
</table>

**Table A.4: Notations for Analysis, Sections 5.2.3 and 5.2.4**

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$CCP^k_j$</td>
<td>A partial class conditional probability of test pattern $x$, given that it has come from class $c_j$, which can be computed at party $P^k$.</td>
</tr>
<tr>
<td>$CCP_j$</td>
<td>The class conditional probability of test pattern $x$, given that it has come from class $c_j$; $CCP_j = \sum_{k=1}^{K} CCP^k_j$.</td>
</tr>
</tbody>
</table>

**Table A.5: Notations for Analysis, Sections 5.2.5 and 5.2.6**

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P^g$</td>
<td>In the vertically partitioned algorithms, this is a party randomly chosen other than $P^c$ and $P^g$ to maintain algorithm security.</td>
</tr>
<tr>
<td>$Y^r_k$</td>
<td>An intermediate distance value calculated in the vertical private algorithms.</td>
</tr>
<tr>
<td>$Y^r$</td>
<td>In the private query algorithm, they represent vectors of shared distance values, indexed by $r$.</td>
</tr>
<tr>
<td>Notation</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>send($P^k, d$)</td>
<td>Sends data $d$ to party $P^k$.</td>
</tr>
<tr>
<td>receive($P^k$)</td>
<td>Returns a piece of data received from party $P^k$.</td>
</tr>
<tr>
<td>broadcast($x$)</td>
<td>Broadcasts a specified piece or vector of data to all parties.  This is executed by all, is filled in by the owner and received by all others.</td>
</tr>
<tr>
<td>$\text{GenerateKeyPair}()$</td>
<td>Generates a pair of keys, one public, one private for use in a public key cryptosystem.</td>
</tr>
<tr>
<td>$\text{Dec}_{sk}$</td>
<td>A decryption function utilizing the private key.</td>
</tr>
<tr>
<td>$\text{Enc}_{pk}$</td>
<td>An encryption function utilizing the public key.</td>
</tr>
<tr>
<td>$\text{secureSum}(P^A, Y^k, Y)$</td>
<td>The function for performing the secure summation across 3 or more parties.  The first parameter is the party where the sum begins and ends, the second is the individual operand, and the third is the variable on $P^A$ where the final result is stored.</td>
</tr>
<tr>
<td>$\text{sharedSecureSum}(P^A, P^B, Y^k, Y)$</td>
<td>Similar to the secure sum, except the final result is securely shared between $P^A$ and $P^B$, with each party placing the final result in their respective variables $Y$.</td>
</tr>
<tr>
<td>$\text{sharedEncryptedSecureSum}(P^A, P^B, Y^k, Y)$</td>
<td>Shares a sum across two parties (the first two parameters).  Uses tree structured computation to calculate the sum with encrypted operands. The third parameter is the operand and the fourth is where the final result is stored.</td>
</tr>
<tr>
<td>$\text{sharedSSP}(x)$</td>
<td>The function which calculates a secure scalar product between two parties, shared randomly between the two calling parties.  The function takes one argument on each party, for the vector involved in the scalar product.  It returns a share on each respective party.</td>
</tr>
<tr>
<td>$\text{SSP}(x)$</td>
<td>Similar to the $\text{sharedSSP}$ function, except the full answer, not a share it sent to party $P^A$.</td>
</tr>
<tr>
<td>$\text{rand}()$</td>
<td>Returns a random integer</td>
</tr>
<tr>
<td>$\text{pad}(Y_r, c_j)$</td>
<td>Pads the members of vector $Y_r$ who are not in class $c_j$ with zeros.</td>
</tr>
<tr>
<td>$\text{randfloat}()$</td>
<td>Returns a random floating point between 0 and 1</td>
</tr>
</tbody>
</table>
APPENDIX B
SMC/PPDM OPERATION EXAMPLES
B.1 Secure Sum

Here an example of the secure sum is presented, described in section 3.1. Suppose there are three parties, $P_1$, $P_2$ and $P_3$, with values 3, 5 and 7 respectively. It is known by all parties that their sum is constrained within a value of 20.

To begin with, $P_1$ generate a random number from $[0, 20]$. Suppose it chooses $R = 12$. It then calculates $12 + 3 \mod 20 = 15$. This value is then passed to $P_2$, which calculates $(15 + 5) \mod 20 = 0$. Now, the value is passed to $P_3$ who calculates $(0 + 7) \mod 20 = 7$. Finally the value is passed back to $P_1$ which then subtracts 12 and reverses the $\mod$ operation to yield 15.

![Diagram of secure sum](image)

**Figure B.1:** Example of the secure sum for 3 parties.

B.2 Secure Scalar Product

Here an example of the secure scalar product (originally developed in [GLL04] and described in this paper in section 3.2) is illustrated in detail. Assume that there are two parties, each with a two-dimensional vector. Party $P^1$ has a vector $(1, 2)$ and Party $P^2$
has a vector $(3, 4)$ (seen in figure B.2). $P^1$ begins by using an additively homomorphic encryption scheme to encrypt its operands, making $(Enc(1), Enc(2))$ and sending these encrypted operands to $P^2$, who can operate on them but not decrypt them.

![Diagram showing the flow of encrypted operands from $P^1$ to $P^2$.]

**Figure B.2:** $P^1$ sends the encrypted operands to $P^2$.

By using the operation which is in the additively homomorphic cryptosystem

$$Enc(A) \otimes Enc(B) = Enc(A + B)$$

and the corollary operation

$$Enc(A)^B = Enc(AB)$$

$P^2$ then calculates:

$$Enc(1(3) + 2(4)) = Enc(1)^3 \cdot Enc(2)^4 = Enc(11)$$

where the power operator is defined within a modular arithmetic described in [Pai99].
$P^2$ then randomly generates a share $s^B = 7$, which becomes its share of the scalar product. $P^2$ then encrypts $-s^B = -7$ and adds it to the scalar product, which yields $P^1$’s share $s^A = 4$ (figure B.3).

Because $P^2$ only has the public key and not the private key for the cryptosystem, it cannot decrypt the encrypted share $s^A$. $P^2$ then passes this encrypted number back to $P^1$, who does own the private key and can decrypt this share.

![Diagram](image)

Figure B.3: $P^2$ performs encrypted operations and sends back the encrypted scalar product to $P^1$.

### B.3 Secure Tree Operations

Here, an example of the tree structured operation $sharedEncryptedSecureSum$ is presented. Tree structured secure operations are described in the document in section 3.3. In the original work discussing tree structured secure operations [VC03b], it is suggested that the traditional secure sum can be adapted to this arrangement if the parties are willing to expose some additional information. However, if the arrangement is augmented with an additively homomorphic cryptosystem, additional security can be obtained.
Suppose there are seven parties (in order to produce a full tree) who want to participate in a tree structured sum. Without loss of generality, they are numbered from $P^1$ to $P^7$ (see figure B.4). The final sum of all of their operands is 28. A party which is a leaf node of the tree must generate a public/private key pair, because if any other party were to generate it, it would be able to immediately decrypt the operand received from one of its children. Without loss of generality, assume that party $P^4$ generates and broadcasts a public encryption key. Again, without loss of generality, assume that the parties who wish to share the final sum are $P^1$ and $P^4$.

![Figure B.4: Leaf parties encrypt their values and send them to their parents.](image)

At the beginning of the encrypted sum $P^4$ and $P^5$ encrypt their operands and send them to $P^2$. $P^6$ and $P^7$ do the same with their operands and send theirs to $P^3$ (illustrated in figure B.4).

$P^2$ and $P^3$ then add their own operands to the two operands that they have received and send these outputs to $P^1$ (figure B.5). $P^1$ then adds both of the received operands to its own operand, giving the final encrypted sum.

Because $P^1$ does not have the private key, it is unable to decrypt the sum it has received. However, it can share this with $P^4$, who does have the key, by subtracting its
Figure B.5: $P^2$ and $P^3$ add their two operands to the ones received.

share as is done in the aforementioned secure scalar product. $P^1$ generates a random number to be its share, uses the cryptosystem to subtract it from the encrypted number, and sends the encrypted remainder to $P^1$, who can then recover its share (figure B.6).

Figure B.6: The sum is shared between parties $P^1$ and $P^4$.

B.4 Public Query, Horizontally Partitioned, Privacy Preserving PNN

Here a detailed example of the horizontally-partitioned, public-query PNN (explained in section 5.2.3) is described. Suppose there are three parties, each of which has one training point of a horizontally partitioned training set for a three-dimensional, two-class
classification problem (class $c_1$ and class $c_2$). The data distribution of this example is illustrated in figure B.7.

$$\begin{array}{|c|c|}
\hline
P^2 & P^3 \\
\hline
(0.3,0.9,0.5): c_2 & (0.6,0.2,0.8): c_1 \\
\hline
\end{array}$$

Figure B.7: Setup of the horizontally partitioned PNN problem

$P^1$ begins by broadcasting a test point $x = (0.6, 1.0, 0.1)$ to all other parties. In order to simplify our calculations, we let $\sigma = \frac{1}{\sqrt{2}}$ over all dimensions, so that in calculating the denominator in the CPDF calculations, $2\sigma^2$, the denominator is $2 \cdot \frac{1}{\sqrt{2}} = 1$. It is assumed that these $\sigma$ values are common knowledge among the parties.

The algorithm then begins to calculate the CCP for class $c_1$. A secure sum is used to calculate the number of training points in class $c_1$ (see section B.1 for an example of the secure sum) which is equal to 2. At each of the parties, the partial CCP value for $c_1$ is calculated. For $P^2$, $CCP^2_1 = 0$ because it does not have any points of class $c_1$. A secure sum is then performed on the partial CCP values, which yields the un-normalized CCP value for class $c_1$ of $CCP_1 = 1.3032$. The process of calculating partial CCPs and adding them by secure sum is repeated for $c_2$ ($CCP_2 = 0.77105$).

Finally, the querying party cycles through each of its available CCP values and normalizes them (dividing by $(2\pi)^{D/2} PT_j \prod_{i=1}^{D} \sigma_i$), yielding $CCP_1 = 0.11702$ and $CCP_2 =$
Figure B.8: Adding the CCP values for \( c_1 \)

0.13847 for \( c_1 \) and \( c_2 \), respectively. Multiplying these by their respective prior probabilities (2/3) and (1/3) for \( c_1 \) and \( c_2 \), respectively, gives values 0.078 and 0.0462 (not actual probabilities due to the simplification of the classifier rule). Class \( c_1 \) is selected for the test point because it maximizes this \textit{a-posteriori}-probability-based value.

### B.5 Private Query, Horizontally Partitioned, Privacy Preserving PNN

In this section, an example of the horizontally-partitioned, private-query PNN (originally explained in section 5.2.4 of the document) is described. Assume the same distribution of data as shown in section B.4, figure B.7: a three-party, two class classification problem, with \( P^1 \) originating a test point query. Again, because \( \sigma = \frac{1}{\sqrt{2}} \), the calculations are simplified because \( x' = x \) and \( X'_r = X_r \).

First a secure sum determines how many points in each class there are in the total training set (see section B.1 for an example of the secure sum). \( P^1 \) begins by communi-
cating with $P^2$ about class $c_1$. However, because there are no points of class $c_1$ in $P^2$, we skip to $P^1$’s communication with $P^3$ to better illustrate our example.

Both parties now know that there are 2 points in class $c_1$ in the total set. To ensure that $P^1$ will not guess how many of those points $P^3$ specifically owns, $P^3$ must have either a real or spurious communication for each one of those points. Therefore, $P^3$ engages in a shared secure scalar product for its own point $X_2^{c_1}$, for which it produces valid shared values, and spurious point, for which it produces spurious values (figure B.9). Following this calculation of the shares of the secure scalar product between the test point and each training point, both parties use the obtained shares to calculate their shares of the distance. Figure B.10 illustrates this calculation for the real point $X_2^{c_1}$.

![Figure B.9: Calculating the SSP between the test point and a training point.](image)

When parties $P^1$ and $P^3$ have shares of each of the values $exp(-dis(x, X_r))$ between the test point and all of the training points, the scalar product of these distances (making sure to skip the spurious values) will yield part of the unnormalized CCP value for class $c_1$. Parties $P^1$ and $P^3$ engage in a shared secure scalar product and add the results to their shares of the CCP value for class $c_1$ (these shares are referred to as $s_{SSP}^A$ and
Figure B.10: Calculating the remainder of the shares between parties.

For the calculation of the unnormalized CCP for class $c_1$ to be complete, $P^1$ must also add the value $\exp(-d(x, X^{1,1}_1))$ for its own training point. The unnormalized CCP value for $c_1$ is finally calculated by secure sum, again yielding 1.3032. After a similar process is repeated for $c_2$, both of these values are normalized as in the example B.4. As before, class $c_1$ is selected to represent the test point.

Figure B.11: The final SSP to yield the CCPs.
Here an example is given for the vertically-partitioned, public-query PNN (originally described in section 5.2.5 of the document). Suppose there are three parties, each of which has a column of a three-dimensional, two-class classification problem. In both of the vertically partitioned examples, the same training set and test point from the horizontally partitioned examples (in B.4 and B.5) is used. Only the distribution of the data is changed. \( P^1, P^2 \) and \( P^3 \) own the first, second and third dimension of the training set, respectively. \( P_3 \) also owns the class labels for the problem. \( P^1 \) issues the test point query. This distribution is illustrated in figure B.12. As in the horizontally partitioned examples, \( \sigma = \frac{1}{\sqrt{2}} \), which simplifies the calculations by allowing \( x' = x \) and \( X'_r = X_r \).

\[
\begin{align*}
\text{p}^2 & \begin{cases} 1.0 \\ 0.9 \\ 0.2 \end{cases} & \text{p}^3 & \begin{cases} 0.2 \\ 0.5 \\ 0.8 \end{cases} \\
& \{c_1\} & & \{c_1\} \\
\text{p}^1 & \begin{cases} 0.7 \\ 0.3 \\ 0.6 \end{cases} & & \\
& x=(0.6,1.0,0.1) \\
\end{align*}
\]

**Figure B.12:** The setup for the vertically partitioned examples.

To begin with, \( P^1 \) broadcasts the query point \( x \) and its public key. For class \( c_1 \) and \( c_2 \), all of the parties participate in secure sum to determine the \textit{a-priori} class frequency \( PT_1 \) and \( PT_2 \), yielding 1 and 2 respectively. These values are broadcast to all parties.
Next, each party calculates the partial distances $Y^k$ from the test point to each training point (figure B.13).

\[ Y^2 = \begin{bmatrix} 0 \\ 0.01 \\ 0.64 \end{bmatrix} \quad \text{and} \quad Y^3 = \begin{bmatrix} 0.01 \\ 0.16 \\ 0.49 \end{bmatrix} \]

\[ Y^1 = \begin{bmatrix} 0.01 \\ 0.09 \\ 0 \end{bmatrix} \]

**Figure B.13: Calculating distance shares for the vertically partitioned public query PNN.**

As shown in figure B.14, the parties then securely sum their distance shares together to produce the full distance values. However, so that no party has knowledge of the full distance values, they are shared between $P^1$ and $P^3$ (the shares are called $s^A$ and $s^B$ in figure B.14).

Now $P^1$ and $P^3$ focus on class $c_1$. To calculate the CCP value for class $c_1$, party $P_1$ and $P_3$ participate in a secure scalar product (figure B.15). However, because $P_3$ knows the classes, and only wants to calculate the CCP for class $c_1$, it will temporarily use zeros for all classes not being considered. These zeros are encrypted for the secure scalar product, so $P^1$ does not know which points belong to which class. The result of this secure scalar product, equal to the unnormalized CCP value for $c_1$ ($CCP_1 = 1.3032$), is now available at party $P^1$. This process is then repeated for $c_2$. 
Finally, as in both horizontally partitioned PNN examples at the querying party, the CCPs are normalized and the final label ($c_1$) is chosen.
Here, an example of the vertically-partitioned, private-query PNN (originally described in section 5.2.6 of the document) is presented. As in the public query, vertically partitioned example (in B.6), there are three parties in a three-dimensional, two-class classification problem. As before, $P^1$ broadcasts its public key. The querying party now encrypts its test point ($x = (0.6, 1.0, 0.1)$) and broadcasts to all other parties. As in the horizontally partitioned examples, $\sigma = \frac{1}{\sqrt{2}}$, simplifying the calculations by allowing $x' = x$ and $X'_r = X_r$. As before, a secure sum is performed to determine the number of points in each class $PT_j$.

Now each party makes an encrypted calculation

$$Enc(Y^k_r) = Enc\left(\sum_{i \in D_k} - (x^{k,j}(i))^2\right) \otimes \left(\prod_{i \in D_k} Enc(x' (i))X^{k,j}(i)\right)^2$$

between the test point and its training points (figure B.16). This operation yields a still encrypted result. Because they are encrypted in a homomorphic cryptosystem, these encrypted shares can be combined by an encrypted secure sum (figure B.17).

Figure B.16: Calculating the encrypted shares for the vertically partitioned, private query PNN.
Figure B.17: Encrypted shared secure sum for the vertically partitioned, private query PNN.

At this point, the algorithm proceeds almost exactly as the vertically partitioned public query, again selecting class $c_1$. 
LIST OF REFERENCES


[KC04b] Murat Kantarcioglu and Chris Clifton. “Privately Computing a Distributed k-nn Classifier.” In 15th European Conference on Machine Learning (ECML) and the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), Pisa, Italy, September 2004.


“System Reliability Theory and Principles Reference from ReliaSoft.”


J. Vaidya and C. Clifton. “Privacy preserving association rule mining in vertically partitioned data.” In In The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 2002.


