APHID: an Architecture for Private, High-performance Integrated Data mining

Jimmy Secretan\textsuperscript{a}, Michael Georgiopoulos\textsuperscript{a}, Anna Koufakou\textsuperscript{a}, Kel Cardona\textsuperscript{b}

\textsuperscript{a}School of Electrical Engineering and Computer Science, University of Central Florida, Orlando, FL 32816
\textsuperscript{b}Dep. of Computer Engineering, University of Puerto Rico, Puerto Rico

Abstract

While the emerging field of privacy preserving data mining (PPDM) will enable many new data mining applications, it suffers from several practical difficulties. PPDM algorithms are challenging to develop and computationally intensive to execute. Developers need convenient abstractions to simplify engineering of PPDM applications. The individual parties involved in the data mining process need a way to bring high-performance, parallel computers to bear on the computationally intensive parts of the PPDM tasks. This paper discusses APHID (Architecture for Private and High-performance Integrated Data mining), a practical architecture and software framework for developing and executing large scale PPDM applications. At one tier, the system supports simplified use of cluster and grid resources, and at another tier, the system abstracts communication for easy PPDM algorithm development. This paper offers a detailed analysis of the challenges in developing PPDM algorithms with existing frameworks, and motivates the design of a new infrastructure based on these challenges.

1. Introduction

Modern organizations manage an unprecedented amount of data, which can be mined to generate valuable knowledge, using several available data mining techniques [1]. 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 potential privacy problems. Organizations, such as health care corporations, 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 dispensing entirely with cooperative data mining, research has instead focused on Privacy Preserving Data Mining (PPDM), which employs 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 a relatively new field, 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 trying to implement PPDM systems would face a lack of available toolkits, libraries, middleware and architectures. The costs involved are potentially high, because of the lack of familiarity with PPDM technology. In addition, 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 integrates high performance computing (HPC) technologies to accelerate the data mining process.

There are several approaches to PPDM, the two most popular of which are data perturbation methods and Secure Multiparty Computation (SMC) methods. Data perturbation methods focus transforming the original data such that it can still be processed by data mining algorithms, while ensuring that the transformed data is difficult to map back to the specific original data. SMC-based privacy preserving data mining involves redesigning data mining algorithms to leverage various cryptographic techniques and private communication schemes. For instance, several SMC-PPDM algorithms make use of an operation called secure sum [2]. A particular PPDM algorithm may be designed such that each party computes intermediate statistics on its own data and then employs the secure sum to combine them.

SMC operations adhere to a clear definition of privacy: the participants in SMC operations should be able to infer no other information than what they can glean using the final output and their own data. SMC methods typically afford greater accuracy and privacy than data perturbation, albeit
at the expense of greater development difficulty and computational requirements. As an example, one study [3] describes building a 408-node decision tree with an SMC-PPDM algorithm from a 1728 item training set in 29 hours. There is much to be done to make these technologies practical.

Although APHID can be extended to implement data perturbation PPDM algorithms, this paper focuses on SMC-PPDM algorithms, because of their increased privacy and accuracy, and because they are the algorithms most likely to benefit from the HPC resources integrated by APHID.

While there is much research that discusses available algorithms and techniques in PPDM ([4, 5, 6, 7, 8, 9, 10, 11, 12] among numerous others), 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 research are: (1) middleware for managing the execution of PPDM algorithms across multiple organizations, (2) the integration of high performance and parallel computing middleware into the PPDM execution environment, and (3) a framework for easily developing PPDM software. First in this paper, a brief background on PPDM (section 2) and high performance computing middleware (section 3) is given. How parties enter into cooperative data mining agreements is briefly discussed in section 4. Then the motivation of APHID is described, along with the development model and the system design (section 5). To illustrate the advantages of APHID and its development model, an SMC-based PPDM algorithm is developed within the framework, the Naïve Bayes classifier (section 6). Performance results and development metrics are then presented for the example algorithm in section 7. Finally, these results are discussed (section 8) and conclusions and future work (section 9) are provided.

2. Distributed and Privacy Preserving Data Mining

Recent research has focused not only the parallel performance of machine learning algorithms, but the problem of coordinating and organizing the huge stream of data that they must process and produce. Often, the totality of an organization’s data assets is stored on multiple sites, with different formats, fields, and extent of records. Much research in data mining up until now has been predicated on the notion that all of the data can be downloaded and coordinated to a single site for processing. Distributed Data Mining (DDM) is the field that challenges this notion, admitting that it may not be feasible to coordinate all of the data centrally, and finding ways around it. DDM is also concerned with managing the distributed and often
heterogeneous computational resources (computing power, transfer bandwidth, connected devices) that the organization’s network has available.

In addition to the issues of security, resource management and algorithm performance which is studied in the area of DDM, privacy adds another difficult design dimension. Privacy Preserving Data Mining (PPDM) concentrates on how to coherently combine and mine databases while preserving the privacy of the individual parties’ data. As a concrete example, suppose numerous different medical research organizations have independent genomic data sets that they wish to mine coherently (e.g. [13]). Because of HIPPA laws, or general apprehension of the organizations to relinquish control of their data, sharing may be restricted. PPDM seeks algorithms and frameworks that will maintain the privacy of the data and summary statistics, while still generating increasingly vital data mining models.

The easiest and most accurate arrangement to ensure privacy in cooperative data mining would be for all of the organizations to transfer their data to a trusted third party, who would consolidate the data on a single machine and execute standard data mining algorithms on the combined data set. Unfortunately, this ideal trusted party may not exist [14, 15, 4, 16, 17, 13]. Even if a suitable third party is found, organizations will likely be apprehensive to relinquish administrative control over their data to a third party. Finally, placing all of this data in a central location may pose an unacceptable security risk, providing one easy point of theft for would-be hackers.

How private cooperative mining is achieved, and the tradeoffs among privacy, accuracy and computational performance differs in the various branches of PPDM. Methods may address individual privacy (e.g. records of specific patients), or collective privacy, that is, the privacy of information about an organization’s records overall (e.g. summary statistics, etc) [18].

First, there are data perturbation methods [19, 20, 21], which focus on obfuscating the original data using random perturbations, while allowing underlying statistical properties to be preserved. A different approach uses Secure Multi-party Computation (SMC) to compute the data mining functions in way that is more exact and private, albeit at the expense of computational efficiency. Because of these distinct advantages, PPDM through SMC is the focus of the proposed architecture. This section explains some of the history and basic concepts in SMC.

SMC has as one of its pillars Yao’s work to solve the Millionaire’s Problem [22]. The Millionaire’s Problem is as follows: two millionaires wish to find 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, sharing random portions of the outputs. Later
it was proved in [23], 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 especially, the computation and communication requirements can make this kind of paradigm impractical for all but the smallest sub problems. This gives rise to more specific solutions using more efficient cryptographic techniques.

This is where certain other cryptographic methods and techniques come to the forefront. 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:

\[ Enc(A) \cdot Enc(B) = Enc(A + B) \]

where \( Enc \) is the encryption function of the cryptosystem and \( \cdot \) is an operation in the cryptosystem. Given this, it is possible for two parties to participate in computation together without compromising their operands. One popular system for homomorphic encryption is provided by Paillier [24]. Cryptosystems such as these have been integrated into larger building blocks in order to implement complex data mining algorithms. A particular secure scalar product algorithm [25] that employs a homomorphic cryptosystem has been used, or suggested for use, in numerous SMC-PPDM algorithms [26, 27, 8, 28]. Another primitive operation that is used in PPDM is the secure sum [2], which allows several parties to add their operands, while revealing only the final sum. SMC-PPDM algorithms typically involve the redesign of the algorithms using SMC techniques such as these.

Most PPDM approaches are posited for cases of either complete horizontal or complete vertical partitioning. 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.

Many popular machine learning algorithms have been recast as into privacy preserving versions. These include decision trees (vertically partitioned in [4] and horizontally partitioned in [29]), the Naïve Bayes classifier (vertical [16] and horizontal [5]), Bayesian networks (vertical [17]), General Clustering (horizontal [6]), k-Means Clustering (vertical [7] and arbitrary [8]), Sup-

While there is certainly a burgeoning interest in PPDM, there are currently few systems to practically support it. Organizations who would want to take advantage of these technologies would be faced with a lack of available support frameworks. There are significant challenges remaining to utilize DDM technologies, and privacy adds an additional level of complexity.

3. Background/Related Work

With the availability of terrabytes of data to be mined, and with more complex machine learning algorithms, high-performance data mining and machine learning have garnered significant attention. 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). APHID seeks to bring both of these areas of technology to bear on the challenge of efficiently building and executing PPDM algorithms. Therefore, this section discusses the available techniques in these areas, and their suitability for PPDM. Finally, some proposed and existing PPDM support architectures are discussed and contrasted with the APHID approach.

3.1. 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 $[36]$ 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. MapReduce has 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 [37].

Other cluster programming environments support less constrained algorithm designs, while still managing execution details. Dryad [38] and Stream Processing Core (SPC) [39] both allow the user to implement functionality in units abstracted as graph nodes. The user also specifies a data flow graph, which is translated into an execution plan by their respective runtimes. While the abstractions provided do not simplify data mining as much as the MapReduce paradigm, they are more flexible and the nodes provide significant modularity.

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 from the power grid. These systems include grids that join supercomputing resources, as well as Networks of Workstations (NOW) architectures, which take advantage of idle desktop machines. 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 [40] 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.

In [41], the authors propose an efficient way of computing data mining algorithms in enterprise grid environments. By leveraging SQL enterprise servers to supply a data set’s sufficient statistics to grid compute servers, the system provides a way distribute processing load, minimize data transfer, and generalize development for various data mining algorithms.

Because these architectures were designed for an environment where data and computation are distributed freely, these architectures cannot be immediately adapted to a PPDM environment, where strict communication and data storage requirements are necessary. However, they can still serve a support role in making PPDM processes more efficient at the intra-organization level. These systems can help reduce both startup and operational costs for getting involved in collaborative data mining.
3.2. Distributed Data Mining

There are several unique design approaches for developing 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 their advantages and disadvantages in the DDM process.

3.2.1. Agent-Based Approaches for DDM

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 from decentralized control, the potential for integrating learning strategies into the agents themselves [42] and simple extensibility by injecting new agents into the environment. In [43], 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 (JAM) system. JAM 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) [44] drives a DDM process between sites with heterogeneous 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 given 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.

The Data Mining Grid Computing Environment (DMGCE) [45] focuses on scheduling of tasks for multi-agent data mining environments. DMGCE splits up data mining workflows on a two tier level, with one stage occurring at a locally administered grid tier, and another stage occurring among independently administered grids. This two tiered arrangement helps to reduce the scheduling complexity. APHID leverages an analogous arrangement with regard to parties in a PPDM process, in order to better manage tradeoffs between privacy and computational performance.

Agent Grid Intelligence Platform (AGrIP) [46] aims to meet several challenges inherent in agent-based DDM, including defining a service architecture, composing services, managing data and developing the agents
themselves. The AGrIP platform leverages MAGE, a multi-agent grid environment supporting development of agents. This toolkit is augmented, by VAStudio, a visual IDE designed to further simplify development.

Despite several successful agent-based systems, there are also potential drawbacks for agents including security and hijacking risks [42]. 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.

3.3. DDM Middleware

The rise of the popularity of grid computing and the release of grid toolkits like Globus [47] has encouraged the creation of middleware to flexibly and extensively support the development and execution of DDM algorithms.

Papyrus [48] 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 [49, 50, 51, 52]. The core layer of K-Grid, implemented on top of Globus services [47], is responsible for managing the metadata about data sources, algorithms and mining tools, as well meeting the requirements of the data mining tools so they are 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.

The Data Mining Grid Architecture (DMGA) [53] describes a collection of services for data discovery, access, filtering and replication, as well as ways to compose services. An implementation of the open DMGA architecture, based on Weka [54] and Globus [47], called WekaG is developed and evaluated on an actual installation and algorithm.

One of the most comprehensive DDM systems is the DataMiningGrid software [55]. It provides functionality for tasks such as data manipulation, resource brokering, and parameter sweeps. It was developed to encourage grid transparency and the adaptation of current code to be grid enabled. The system aims to be accessible by both experts and novices with experts able to graphically describe a detailed workflow, and novices able to submit simple requests through a web based system. The authors emphasize that
extensibility is 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.

While systems like Papyrus, K-Grid, DMGA and DataMiningGrid will no doubt become an important part of future DDM applications, they are not designed for the unique needs of PPDM algorithms. They do not necessarily provide the facilities to express the strict organizational separations necessary in PPDM. Furthermore, these systems tend to focus on the broad data mining process (e.g. workflow, data management, etc.) and not on providing interfaces for coherent distributed algorithms.

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 [56] argues that P2P networks can offer the following to DDM: scalability, availability, asynchronism and decentralization. The DHAI system [57] 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 participate 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. In addition, some of the principle advantages of P2P networks, including fault tolerance and load balancing can be difficult to translate to a PPDM environment. For instance, suppose the host storing a portion of a database being mined goes down or is being intensively processed. This portion of the data cannot be arbitrarily replicated in the network, because it is private. In addition, in the typical PPDM suggested in this paper will involve from a few to tens of organizations. Substantial agreements must be made to include participants in these PPDM consortia: parties are not simply expected to enter and leave the mining operations at will.

3.3.1. 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 technically beneficial but also commercially beneficial [58].

WebDisC is a web based system for meta-learning [59]. 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. While WebDisC works well with classifiers independently trained at parties, it is not designed for a single, coherent PPDM algorithm.

The Webservices Oriented Data Mining in Knowledge Architecture, or WODKA [60] provides a robust Enterprise Service Bus (ESB) to support service oriented interaction with grid data mining software. It is specifically designed to work with previously developed grid-based Association Rule Mining software called DisDaMin [61]. WODKA’s ESB approach to SOA supports improved reliability and uniformity for the exposed interfaces.

Service composition also has the potential to support complex DDM applications. The system in [62] 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.

It is clear that services have the potential to integrate DDM processes across heterogeneous systems and environments because of widely adopted services standards, and the fact the services frameworks are available for numerous languages and environments. It is for this reason that APHID and much of the DDM middleware in the previous section are based off a service oriented architecture (SOA). While the existing systems surveyed were not specifically geared toward the development of large-scale, coherent PPDM algorithms, they provide the glue to connect the architecture.

3.4. Architectures to Support PPDM

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 [63] proposes a runtime environment to support privacy preserving data access and mining. Built on top of a service oriented architecture and communicating over JXTA peer-to-peer technology [64], TRIUMF aims to provide 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. While TRIUMF can potentially enable efficient PPDM processes, and scale to many parties,
it does not suggest a clear framework with which to implement a variety of PPDM algorithms.

In [65] 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 VOs. 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. The system described in [66] suggests improvements to the Globus GSI security framework [47], such as sandboxing and intrusion detection, to support PPDM. It also suggests that SMC and randomization services be built on top of the K-Grid architecture [49, 50, 51, 52] for DDM. While APHID does follow a similar technique of providing SMC services, it again differs from the work in [66] by suggesting a coherent and concrete programming framework in which to develop PPDM algorithms.

Some systems are built specifically on an automated framework of privacy preservation. Fairplay [67] 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 [68], 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 and libraries. 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.

4. PPDM Preliminaries

To design a system that serves the needs of organizations cooperatively employing PPDM, it is beneficial to first describe how such a data mining cooperative would look. The PPDM arrangement will include two or more parties, distinct organizations for which sharing data is undesirable for legal or competitive reasons. Before entering into a data mining cooperative, it is likely that the organizations would enter legal agreements reinforcing their commitment to participate in the PPDM process in good faith, and not to engage in malicious behavior or network intrusion.
As a result of legal reinforcements, the parties in many PPDM cooperatives would be called semi-honest in PPDM terminology. That is, they do not engage in malicious communication or hacking to disrupt the network, but instead could be tempted to leverage information received from other parties to find out as much as possible about other parties’ databases. While many PPDM algorithms are design under this assumption, they may also be designed to resist malicious parties. This level of design is particular to each algorithm, with APHID supporting both. It should be noted 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 [23].

From the technical standpoint, each party may have different administrative policies, firewalls, intranets, and computing resources. An organization representing a party may have on the order of hundred or even thousands of PCs, several mainframes or computing clusters, and other dedicated computing hardware. Parties are connected to each other through the Internet, through links that are low-bandwidth relative to their internal networks (e.g. T1 lines). They will often be geographically quite far away from each other.

APHID does not address general network security within organizations. The exposed APHID services and access to the database are presumed to be secure. That is, all of the necessary firewalls, password protection, security patches are in place. Accomplishing this security is no small feat, and is the topic of much orthogonal research.

5. 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. Secondly, organizations would need a middleware framework to support PPDM, which is not sufficiently provided in current systems [48, 50, 55]. 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 identified in the literature or found lacking in other systems. The system must have:
1. Low development cost [69] (by mitigating development complexity and encouraging reuse)
2. A runtime environment for executing algorithms (as in [48, 50, 55, 63, 66])
3. The ability to leverage high-performance computing resources (such as [36, 40] and others)
4. Flexibility to support many PPDM algorithms [70]
5. Support many popular languages (i.e. portability [69])
6. Scaling to support numerous parties and users [69]

To support low development cost and language independence, DDM/P-PDM functions are provided as a collection of web services (as suggested in [50, 55, 63, 66]), 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. They 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 5.2. MEL orchestrates the execution of the PPDM algorithm, providing the necessary runtime environment.

APHID is explicitly built on a two-tier system of PPDM (analogous to the use of two different grid tiers of task scheduling used in [45]) which differentiates it from other PPDM 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 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 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 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. 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
Figure 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.

and transferred to a high-performance distributed file system (e.g. HDFS [71], 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 manages privacy preserving interactions with external parties. 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 accesses
training databases, and submits 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 [69], 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 an application. 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, the HPC operations can be implemented on a simple server.

Figure 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 PPDM process. The MEL 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 PPDM services, which communicate with other parties. As data moves from right to left (i.e. from outside servers to local clusters), faster interconnects tend to be found (e.g. from T1 lines communicating to PPDM services to fast fiber channel lines connecting cluster nodes). For this reason, data can be cached at each layer.
First, the development model around which APHID is structured is described. Then each of the three layers is described in detail.

5.1. 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. The cornerstones of APHID development are a program style similar to many HPC frameworks, and policy-attached shared variables, which mitigate complexity and cost (item 1 of our desiderata). Research in parallel and distributed data mining has sought to ease development by introducing new simpler development models [72, 36, 73], and APHID continues in this spirit.

5.1.1. 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) [74], 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.

5.1.2. Shared Variables

One technique APHID employs to simplify PPDM application development is the use of a special type of shared variables. The shared variables follow a publish-subscribe model, with an attached policy. A policy determines how and by whom the value may be accessed. The shared variable has one party that can write (publish) values to the variable, and one or more parties who can read it. Parties subscribe to the variable and will receive notification when the value has changed.

Every policy defines a sharing type. The available policy sharing types are shown in table 1. The first type of sharing, Intra-Party (IP) sharing 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 sharing type, Fully Shared (FS), creates variables for which the unmodified value can be passed among parties. Finally the Secret Shared (SS) type 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 [25]. 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 simplest policies define an access list of parties for which the specified type of sharing is permitted.

Table 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>

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 3 with language-neutral pseudo-code. Line 1 declares a shared variable, and line 2 attaches a policy: in this case, the $V_{\text{shared}}$ is fully shared between party $P_1$ (who can read and write its value) and party $P_2$ (who can read its value). In lines 3–4, executed by party $P_1$, a value of 1 is written to $V_{\text{shared}}$. When in lines 3–4, the value of $V_{\text{shared}}$ is read by $P_2$, $P_2$ automatically requests and caches that value.

Shared variables with policies offer automatic checking for authorization. Figure 3 also gives an example of this scenario. If code is accidentally written such that it tries to access a variable locked onto another party (lines 7–8), the runtime can prevent access. Upon trying to retrieve this value, the runtime will throw a security exception, and the execution will typically be halted.

On lines 9–10, the program declares an intra-party variable that can be easily passed among layers. For instance, $X$ may be a vector that is currently cached on a layer other than the one where the listed program is executed. In lines 11–12, a secret-shared variable is declared, that stores the
result of a shared scalar product (lines 13–14). The shared scalar product takes a different vector from each party, and returns to each party a random share of the scalar product. Therefore, the value of $S_{shared}$ is different on $P^1$ and $P^2$, and unavailable on any of the other parties.

```plaintext
float $V_{shared}$;
setPolicy($V_{shared}, Policy_{FS}(P^1, P^2)$);
if $P == P^1$ then
  $V_{shared} = 1$;
else if $P == P^2$ then
  $R = 1 + V_{shared}$;
else if $P == P^3$ then
  $R = 2V_{shared}$;
end
Vector $X$;
setPolicy($X, Policy_{IP}$);
int $S_{shared}$;
setPolicy($S_{shared}, Policy_{SS}(P^1, P^2)$);
if $P == P^1 || P == P^2$ then
  $S_{shared} = SSP(P^1, P^2, X)$;
end
```

Figure 3: Examples of shared variable usage

Policies can also serve more complex access control functions. One potential source of information leak is if too many queries or operations are performed on certain data, which has the potential to allow parties to characterize another party’s database beyond acceptable privacy limits. Fortunately, even if the operation in question is built into several different algorithms (e.g. a secure sum occurring in both Naive Bayes and decision tree classifiers) these policies can catch this violation through checking access records on permanent, system-wide database storage. These policies can also be used to filter final results and data mining models that would violate privacy. As previously stated, even though an algorithm meets SMC criteria, its final results may still violate privacy. By attaching policies to the final classification or model data structures, users can add safeguards against revealing information through the model.

Figure 4 demonstrates some of these complex, user-defined policies. Sup-
pose that a particular secure scalar product should only be performed a limited number of times before it supplies too much information to the other party. In line 2, a policy is attached to a transaction variable $T$ that limits how frequently the scalar product can be called on a particular value, by a particular party. Per the APHID system settings, APHID will store in a database a record of when a particular secure operation was called, and with what data. If the secure scalar product called on line 6 will violate the established limits, it will generate an error. The variable $FrequentItems$ is declared to hold the frequent items found in total dataset. It has two policies attached: a simple policy declaring it fully shared between the two parties (line 4), and a complex, user defined policy (line 5) that checks for potential inverse frequent item set mining violations (see [75] for more information). As new candidates are added to the frequent item set by each party, this policy is engaged and will generate an error and halt being read when a candidate has been added that can potentially lead to privacy violations.

1 Vector $C$
2 setPolicy($C$.LimitOccurance(ScalarProduct));
3 Set $FrequentItems$;
4 setPolicy($FrequentItems$.PolicyFS($P_1$, $P_2$));
5 setPolicy($FrequentItems$.InverseFIMChecker);
6 if ScalarProduct($P_1$, $P_2$, $C$) > MinSup then
7       $FrequentItems = FrequentItems \cup C$;
8 end

Figure 4: Examples of complex, user-defined shared variable policies.

5.2. Main Execution Layer

The Main Execution Layer (MEL) is a collection of services that compromise full data mining algorithms (e.g. Naïve Bayes, k-NN, ARM, etc.), which are then in turn 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. First, a user within one of the parties (or external to the party, if they have proper access) calls a web service to either submit a query or start building a model. The model can include a trained classifier, clusters, association rules, etc. Then, the party initiating the query establishes a unique session ID, and registers this session ID will all of the involved
parties. The PPDM algorithm then calls operations on its data with HPC services and perform SMC operations through its PPDM 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 which stores metadata about data sets to be mined and information about the execution of PPDM algorithms. The DMED also stores metadata associated with complex shared variable policies. For instance, it may store the number of times that a certain SMC operation has been performed on a specific dataset, to make sure that data privacy is not compromised from overuse.

5.3. High-Performance Computing Services

With the astronomical size of data sets available for data mining, high performance computing resources are necessary to make the mining process practical [1]. Data mining systems must also integrate closely with an organization’s databases, without disturbing the organization’s everyday business processes. An organization typically has one or more databases storing data such patient records, scientific or manufacturing observations or customer transactions. 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. 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 the existence 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 [74] can be wrapped in these services, using automated HPC frameworks can significantly simplify development at this layer. Systems like MapReduce [36], Dryad and Distributed DOALL [40], are well suited to simplifying the development of large-scale database operations. For the implementation of APHID discussed in this paper, a MapReduce system called Hadoop [71] supports the database processing operations.
Within an organization’s APHID installation, the databases to be mined are periodically transferred and synchronized to the distributed filesystem used by Hadoop, HDFS [71]. There is not necessarily a one to one mapping from business databases 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 to the DMED, along with its metainformation (data provenance, intended purpose, etc).

5.4. PPDM Services

The PPDM Services (PPDM-WS) is responsible for both providing primitive SMC operations (e.g. secure sum), and for providing the send, receive and peer finding services on which those operations are built. Developing this set of services for PPDM is efficient because most popular SMC-based PPDM algorithms utilize a small set of SMC operations. By providing a toolkit of frequently used operations, as suggested in [76], developers can easily implement numerous PPDM algorithms. Table 2 lists algorithms which utilize popular SMC operations.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secure Sum [2]</td>
<td>[77, 12]</td>
</tr>
<tr>
<td>Secure Scalar Product [25]</td>
<td>[26, 27, 8, 28, 12]</td>
</tr>
<tr>
<td>Yao Circuits [22]</td>
<td>[26, 14, 8]</td>
</tr>
<tr>
<td>Oblivious Transfer [78]</td>
<td>[5, 29]</td>
</tr>
<tr>
<td>Secure Matrix Products [79]</td>
<td>[80]</td>
</tr>
</tbody>
</table>

5.5. Data Management Between Layers and Parties

Breaking the execution of a PPDM algorithm into three layers helps to modularize development, and allows the framework to scale. However, this arrangement can cause performance problems when large amounts of 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, resulting in $10^7 \times 512 = 5GB$ of data. If the data must be transferred from
the HPC-WS layer, to the MEL, to the PPDM-WS layer, and finally to another party, the operation could become a significant bottleneck.

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 development, and runs contrary to APHID’s design philosophy. Therefore, APHID abstracts the details of the transfer from the developer. When APHID’s services process a computation, a reference handle 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. If the layer does not need direct access to the data in that handle, it can pass it to another layer or party by passing only the handle.

Each of the three APHID layers has its own cache. If caching is enabled, APHID follows a write-invalidate policy: when the publisher writes to its shared variable, it sends an invalidate message to all known requesters of the variable. 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.

6. Example Algorithm Implementation

A reference APHID architecture has been 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 [71], which includes support for Java, C++ and Python.

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 this section, the privacy preserving Naïve Bayes algorithm for horizontally partitioned datasets [5]. The privacy preserving Naïve Bayes is well suited to take advantage of HPC resources, because of an intensive database processing step and light SMC operation requirements. While this is a classification algorithm, 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. Before introducing these algorithms, some standard notation is first established.

6.1. Notation

During a PPDM algorithm execution there are $K$ parties involved, numbered $P^1,...,P^K$. In the case of horizontally partitioned classification, the
D-dimensional training data, $X$, are divided among the parties such that each party $P^k$ owns several $D$-dimensional instances of the training data. There are a total of $N$ training points in the total set $X$. An arbitrary point in the training set is denoted as $x$, and a test point is denoted as $x_{test}$. The test point and all of the training points are members of one of the $J$ classes in $c$, $c_1$ to $c_J$.

6.2. Example: Privacy Preserving Naïve Bayes Classifier

By assuming that the attribute values of the data are conditionally independent, the Naïve Bayes employs the following simplified classification rule:

$$c_{NB}(x_{test}) = \arg\max_{c_j \in c} \left( P(c_j) \prod_i P(x(i) = x_{test}(i) | c_j) \right)$$

That is, the test point $x_{test}$ belongs to the class $c_j$ which maximizes the given equation (which is directly related to the a-posteriori probability). The value of $P(c_j)$ is estimated by $\frac{N_j}{N}$, that is, the number of training points of class $c_j$ divided by the total number of training points.

The value $P(x(i) = x_{test}(i) | c_j)$ represents the probability that the $i$th attribute of members of the dataset of class $c_j$ will equal $x_{test}(i)$. For categorical attributes, $P(x(i) = x_{test}(i) | c_j) = \frac{N(x_{test}(i), i)}{N_j}$, where $N(v, i)$ represents the number of training points in $X$, whose $i$th attribute has the value $v$. Without loss of generality, only categorical attributes are dealt with in our example implementation.

Because Naïve Bayes is popular and well explored in the field of data mining, a more detailed explanation is deferred to the literature [81, 82]. Instead, we now discuss how the privacy preserving Naïve Bayes algorithm is implemented in the APHID framework, with details for each of the three service layers. The privacy preserving Naïve Bayes algorithm is similar to the standard algorithm, with the addition of secure sum operations to add class frequencies and attribute-value-class counts across all parties. Figure 5 shows the algorithm executed at the MEL.

To begin with, on line 4, the main service at the MEL calls a service at the HPC layer called `sumElementsByCategory`. This function counts the number of instances for each attribute-value-class combination in the dataset and returns the handle to the array of counts $s_k$. Note that because $s_k$ is returned as a handle from the HPC services layer, it can be directly transferred to the PPDM services layer to become an operand for the secure sum, without passing through the MEL. The counts from all parties are added together through secure sum from the PPDM layer, and stored at
public class NBayesHoriz extends PPDMAlgorithm {
  public void createModel(TrainingSet tr) {
    Handle s_k = HPCServices.sumElementsByCategory(tr);
    Handle s_y = PPDMServices.secureSum(config.getParties().firstElement(), s_k);
    Handle N_j = PPDMServices.secureSum(config.getParties().firstElement(), tr.getClassFrequencies());
    if (config.getMyParty().equals(config.getParties().firstElement())) {
      Integer[] N_j_Array = N_j.toValue(Integer[].class);
      Double[] Pc = new Double[N_j_Array.length];
      Integer[] s_y_Array = s_y.toValue(Integer[].class);
      Double[] Px = new Double[s_y_Array.length];
      for (int i = 0; i < training.getUniqueCategories(); i++) {
        Pc[i] = (double) N_j_Array[i] / (double) tr.getTotalGlobalInstances();
        Px[i] = s_y_Array[i] / tr.getClassFrequencies()[i];
      }
      model = new NBModel();
      model.setAprioriProbs(Pc);
      model.setCategoryProbs(Px);
    }
  }
}

Figure 5: A Java implementation within APHID of the main Naïve Bayes service, executed in the MEL.
the first party in $s_y$ (lines 6–8). An illustration of the data flow involved in these lines can be seen in figure 6. In lines 10–12, the parties also add the number of instances in each class through a secure sum. Lines 14–33 are executed only on the first party where both the summed counts currently reside. On line 17, the Handle for $N_j$ returns the array value that it wraps. If it has not already done so, the MEL layer of the first party automatically requests this array from the PPDM layer. Something similar happens with the $s_y$ handle on line 20. The elements in $N_j$ are divided by the total number of instances, and the elements of $s_y$ are divided by the number of instances in each class to respectively produce the apriori and class conditional probabilities, completing the Naïve Bayes classifier model.

![Diagram](image.png)

Figure 6: Shared variable handles avoid unnecessary transfer. This illustration follows lines 4–8 of the code for the NB classifier. Within party 1, the MEL requests the category sum computation, which is computed on HPC resources, and stored at the HPC-WS layer. A handle is returned for the data, which is passed through the secure sum to the PPDM-WS layer. This handle reaches the MEL of party 2, which requests the actual data and performs a sum on it. Normal weight lines represent lightweight requests or handle transfers, and heavier lines represent substantial data transfer.

In the algorithm in figure 5, the primary mechanism for secure computation is the secure sum [2]. We now discuss its implementation in the PPDM service layer.
6.2.1. Secure Sum

As mentioned before, a fundamental algorithm in SMC is the secure sum \([2]\), which 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 value to \(P^2\). The modulus by \(F\) is necessary to ensure that the sum is kept randomly distributed within the range \([0...F]\), yielding no additional information to any observer. \(P^2\) takes this and then calculates \(((v^1 + R) \mod F + v^2) \mod F\), sending this to \(P^3\). Parties \(P^3\) through \(P^K\) add their operand to the sum they have received and then send it to the next party. 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 \(v = \sum_{k=1}^{K} v^k\).

![Diagram of the secure sum for K parties. The first party adds a random number to its operand and transmits it to the next party. Each party then receives the sum from the previous party in the ring, adds its own operand, and transmits it in the same way. Finally, when the sum reaches the originating party, the random value is subtracted, revealing the sum.](image)

Figure 8 gives the Java implementation of the secure sum service, used by the example Naïve Bayes algorithm. It should be noted that developers will typically use a built in library of SMC operations, and should rarely have to implement their own. The service takes as arguments the base party where the final result should end up, and a handle wrapping an array of doubles, coming from each individual party calling the function. To begin with, the function converts the \(myOperands\) handle to its array of doubles (line 6). 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 8–9, 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 by 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 11–33 are executed only on the base party, and the rest is executed on the other parties. In lines 17–21, a random vector is generated and added to the operands at the base party. The resulting vector 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 (lines 39–41) and sends them to the next party (line 42). This token finally reaches back to the base party, at line 25, which subtracts the random elements to produce the final sum. The result is stored in the PPDM layer (through the call on line 32), and becomes available to any system within the base party’s stack. It should be noted that data can be transferred both automatically through shared variable management, as well as explicitly through send and receives. These complementary operations are implemented through similar request mechanisms.

6.2.2. Calculating Frequency of Attributes

The algorithm given in figure 9 represents a portion of the Naïve Bayes algorithm which executes in the HPC layer. Because we are not concerned with the details of the MapReduce implementation, and in the interest of brevity, pseudo-code rather than Java code is given. This MapReduce program executes on a cluster with the training database $X$ distributed in cluster storage. The map procedure in lines 1–4 takes each training point as the value, and its index $r$ as the key. The map phase breaks each training point down into an attribute-value-class combination across each dimension. In the reduce phase (lines 6–8), these are summed together into a hashtable responsible for keeping the counts of of attribute-value-class combinations. A vector of counts associated with each combination of dimension $i$, the dimension’s possible values, and the possible class values $c_j$ is returned to the calling application within the MEL.

Access to this MapReduce program is wrapped in a simple service (not shown) that passes it a reference to the training set on which it is run, and submits the job to the MapReduce master server. For MapReduce programs that rely on data from other layers or parties, the HPC services layer can cache the necessary data and distribute it to the compute servers before the MapReduce computation begins.
```java
public Handle secureSum(Party baseParty, Handle myOperands) {
    Party nextParty = PPDMConfig.getInstance().nextParty();
    Party previousParty = PPDMConfig.getInstance().previousParty();

    Double[] myOpsArray = myOperands.toValue(Double[].class);
    Handle toRet = 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 < myOpsArray.length; i++) {
            randVec.add(random.nextDouble() * RANDOM_LIMIT);
            myOpsArray[i] = (myOpsArray[i] + randVec.lastElement())
            % RANDOM_LIMIT;
        }
        send(nextParty, myOpsArray);

        Double[] recvd = receive(previousParty, Double[].class);

        for (int i = 0; i < recvd.length; i++) {
            myOpsArray[i] = (myOpsArray[i] - randVec.elementAt(i))
            % RANDOM_LIMIT;
        }
        toRet.putValue(myOpsArray);
    } else {
        Double[] recvd = receive(previousParty, Double[].class);
        Double[] arrayToSend = new Double[myOpsArray.length];

        for (int i = 0; i < recvd.length; i++) {
            arrayToSend[i] = myOpsArray[i] + recvd[i];
        }
        send(nextParty, arrayToSend);
    }
    return toRet;
}
```

Figure 8: A Java implementation within APHID of the secure sum algorithm, executed at the PPDM services layer.
7. Performance Results

In order to evaluate the performance of the framework, an artificially generated classification training set was executing using the implemented Naïve Bayes classifier. The test set contained 64 dimensional categorical data, generated with a performance testing tool [83]. All tests were executed on a single cluster with nodes containing dual 2.2GHz Opteron processors, and 3GB of RAM, connected by gigabit ethernet. Parties were simulated by logically grouping together nodes of the cluster. While the interconnect used in testing was certainly faster than will be available for most inter-organizational connections, the effect on the timing of the Naïve Bayes should be minimal. The only communication on wide-area networks that would occur in actual execution of the algorithm would involve four HTTP transfers per party (two per secure sum, with two in the algorithm) amounting to only a few kilobytes each.

In the first set of tests, the number of parties was varied from three to eight. The three layers of the stack, the Hadoop master server and a Hadoop compute node were all run within a single server node. Each party was given a small training set with 8,000 instances. The graph in figure 10 demonstrates that the APHID runtime system can operate efficiently, with acceptable overhead, taking on the order of tens of seconds. The time also scales modestly with the addition of parties.

When the mining task becomes larger and more computationally demanding, APHID can scale to meet those demands, as evidenced in figure 11. Figure 11 demonstrates how well the evaluated PPDM algorithm was
able to take advantage of cluster resources. For these tests, the number of parties involved was fixed at three. For each party, the number of associated cluster nodes was varied from 1 to 16. Because one additional cluster node is used to run the APHID stack per party, the total number of servers in each run varied from 6 to 51 ($3 \times (1 + 16)$). Three separate test were conducted, giving each party 2 million, 5 million and 10 millions points (for a total of 6 million, 15 million, and 30 million points respectively). 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 to create the Naïve Bayes classifier model. In the run with 10 million points per party, the additional nodes were able to reduce the runtime from over 30 minutes to a little over 2 minutes, which could have a significant impact for time-sensitive applications. For more intensive algorithms, this difference can tremendously increase the practicality of PPDM.

8. Discussion

By utilizing existing cluster and grid infrastructure within each organization, the additional hardware required for the framework is minimal, and
can be scaled with increased load. Each organization would only require one additional server to support the framework. This server would host the local grid services, the data mining applications and the DDM/PPDM services. These three layers can also be split onto additional servers as load increases.

APHID successfully integrates the use of parallel and high performance computing resources, without significantly increasing the development complexity for PPDM algorithms. The use of the APHID development framework comes at relatively little cost in overhead and can immediately facilitate practical PPDM applications. APHID’s extensibility should allow it support newer, more advanced SMC techniques as they become available.

9. Conclusions and Future Work

Future work will include the development of more SMC and grid mining primitives to ease development of new algorithms. 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 PPDM services layer.
Communication efficiency is also a significant challenge with web services architectures. While these communication paradigms allow unprecedented interoperability, they can perform slowly [84]. Future research will lead to 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 [85] and Pig [86] offer high level interfaces to querying and processing data, which further abstract and simplify the process of parallel development. Such advances can easily be integrated into the developed architecture.

It is our sincere hope that APHID and its future extensions will facilitate a new generation of PPDM applications, and highlight the potential for responsible, private data mining that is still capable of distilling vital knowledge.

10. Acknowledgment

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, 0806931, 0837332.

References


35


