

Minimizing Cost when using Globally Distributed Cloud Services: A Case Study in Analysis of Intrusion Detection Workflow Application

Suraj Pandey¹, Kapil Kumar Gupta¹, Adam Barker², Rajkumar Buyya¹

¹The Cloud Computing and Distributed Systems Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Australia
{spandey, kgupta, raj}@csse.unimelb.edu.au

²Department of Engineering Science
University of Oxford, UK
adam.barker@gmail.com

Abstract

*Cloud computing is an emerging technology that allows users to use computation, storage and data access services from around the world. However, Cloud service providers charge users for using these services. Specifically, to access data from their globally distributed storage edge servers, they charge users depending on the user's location and the amount of data transferred. When moving data-intensive application to Clouds for scalability, optimizing the cost of transferring data to and from these edge servers becomes the 'first objective', as data play the dominant role in the application's execution. In this paper, we formulate a non-linear programming model to minimize the data retrieval and execution cost of data-intensive workflows on Clouds. The model retrieves data from Cloud storage resources such that the amount of data transferred is inversely proportional to the communication cost from those resources to a compute resource where a task is assigned for execution. We take an example of an 'intrusion detection' application workflow, where the data logs are made available from globally distributed Cloud storage servers. We construct the application as a workflow and experiment with Cloud based storage and compute resources. We compare the cost of multiple executions of the workflow given by a solution of our non-linear program against that given by Amazon CloudFront's 'nearest' single data source selection. Our results show a savings of **three-quarters of total cost** using our model, however with an acceptable compromise in application computation time.*

1 Introduction

Increasingly, scientific and commercial applications are leveraging the power of distributed computing and storage resources [4, 15]. These resources are available either as part of general purpose computing infrastructure such as Clusters and Grids, or through commercially hosted services such as Clouds [1]. Buyya et al. [3] have defined Clouds to be a type of parallel and distributed system consisting of inter-connected and virtualized comput-

ers. These computers can be dynamically provisioned as per users' requirements. Thus, to achieve better performance and scalability, applications could be managed using commercial services provided by Clouds, such as Amazon AWS, Google AppEngine, and Microsoft Azure. However, the cost of computing, storage and communication over these resources could be overwhelming for compute-intensive and data-intensive applications.

Data mining is one example application domain that comprises of data-intensive applications often with large distributed data and compute-intensive tasks to manage data and knowledge distribution. Examples of data mining applications are: checking bank account lists with lists of suspected criminals (Watch List Compliance), checking for duplication of customer data in financial marketing, using catalogue data in astrophysical image analysis or detecting the spread of Internet worms using intrusion detection systems. The data to be mined may be widely distributed depending on the nature of the application. As the size of these data-sets increases over time, the analysis of distributed data-sets on computing resources for multiple users (repeated executions) has the following challenges:

- Large number of data-sets and mining tasks make the application complex → requires a well-designed application workflow
- Large size and number of distributed data-sets make the application data-intensive → requires minimization of communication and storage costs
- Cost of computing (classification/knowledge discovery) and transferring of data increases as the number of iterations/data-sets increase → requires minimization of repeated data mining costs

In this paper, we address the challenges listed above for data-intensive workflows by making the following three contributions:

1. We model the cost of execution of a workflow on Cloud resources using a Non-Linear Programming (NLP) model. The NLP-model retrieves data partially

from multiple data sources based on the cost of transferring data from those sources to a compute resource, so that the total cost of data-transfer and computation cost on that compute resource is minimized.

2. We take Intrusion detection as a data mining application for a case study. This application has all the features as listed in the previous paragraph when executing commercially [15]. We design the application as a workflow that simplifies the basic steps of data mining into blocks.
3. We then apply the NLP-model on the intrusion detection application to minimize repeated execution costs when using commercial compute and storage resources. We compare the cost when using Amazon CloudFront with simple round-robin scheduling algorithm.

The remainder of the paper is organized as follows: we present related work in Section 2; intrusion detection application and its workflow design in Section 3; cost minimization problem using NLP model in Section 4; the NLP-model and its use for the intrusion detection application in Section 5; experimental setup in Section 6 and analysis in Section 7. With future work, we conclude the paper in Section 8.

2 Related Work

Armburst et al. [1] described the benefits of moving to Cloud computing. These benefits include lower operating costs, physical space savings, energy savings and increased availability.

Deelman et al. [4] presented a case study for examining the cost-performance tradeoffs of different workflow execution modes and provisioning plans for Cloud resources. They concluded that data-intensive applications can be executed cost-effectively using Cloud computing infrastructure. In our paper, we focus on the minimization of communication cost using globally distributed Cloud edge-servers and compute nodes.

Amazon CloudFront¹ uses edge locations in United States, Europe, and Asia to cache copies of the content for faster delivery to end users. It provides users address in the form of a HTTP/HTTPS uniform resource locator (URL). When a user requests one of these data from any site, Amazon CloudFront decides which edge location is ‘best’ able to serve the request to that user’s location. However, users do not have control over the amount of data to get from each edge servers, to minimize cost, unless they access the URL from a different location. We compare our approach with this ‘best’ location approach.

Broberg et al. [2] introduced MetaCDN, which uses ‘Storage Cloud’ resources to deliver content to content creators at low cost but with high performance (in terms of

¹<http://aws.amazon.com/cloudfront/>

throughput and response time).

Microsoft has a Windows Workflow Foundation for defining, managing and executing workflow as part of its .NET services. With the .NET services, workflows can be hosted on Clouds for users to access it from anywhere [9]. The service facilitates transparent scalability for persistence stores and distribution of load between hosts.

A number of work in Grid computing, especially those related to Data Grids, have focused on optimal selection of data sources while scheduling applications [12, 8]. In Grids, users were not able to provision specific type of resources at specified locations as per application requirements. In Clouds, however, users can first choose the set of compute and storage resources they want for their application and then use our model for minimizing the total cost. The initial selection may be based on user’s budget allocated for executing the application in Clouds.

Wu et al. [14] presented the design and implementation of Collaborative Intrusion Detection System (CIDS) for efficient intrusion detection in a distributed system. They claim that aggregate information is more accurate than elementary data for intrusion detection.

Zeller et al. [15] presented the advantages of using Cloud computing for data mining applications, especially when the size of data is huge and globally distributed.

3 Intrusion Detection Using Data from Distributed Data Sources

First, we describe an use-case for Internet worm detection. Then, we describe the process of intrusion detection in general and present a workflow design for executing data mining steps over distributed intrusion data logs.

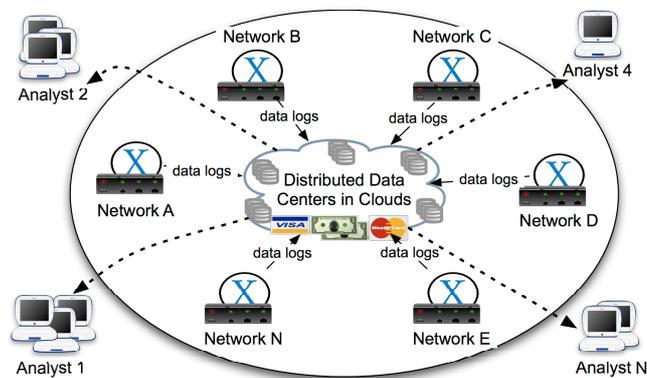


Figure 1: Global Intrusion Detection scenario

3.1 Intrusion detection scenario

We take an example of detecting the spread of a malicious worm over the Internet. In practice, a large number of independent networks spanning throughout the Internet share their network logs to detect such an outbreak. The logs from each individual network are continuously fed to

the Amazon Cloud storage (or some other services), which distributes them to globally distributed edge servers.

The aim of the intrusion detection system (or the analyst) is to analyze these combined logs to detect an outbreak of a worm. Such analysts can be located at multiple locations close to some of the data sources but at a large network distance from a majority of the other data sources.

Assuming that every intrusion detection system (or analyst) follows the same data mining process, which we describe later in the paper, the Naive approach is to separately aggregate the log data from all independent networks for every analyst. It is not hard to visualize the redundancy in the data transfer (for each individual network) and hence the cost associated with such massive amounts of data transfer.

Using the distributed edge servers, we can minimize the cost of data transfer to each individual intrusion detection system (analyst). We represent this scenario in Figure 1. With an aim to minimize the cost of data transfer, we develop a non linear programming based approach, described later in the paper, and compare it with the standard nearest source approach adopted by CloudFront and observe that our model achieves a significant savings of three-quarters of the total cost.

3.2 Intrusion detection process

Intrusion detection as defined by the SysAdmin, Audit, Networking, and Security (SANS²) institute is the act of detecting actions that attempt to compromise the confidentiality, integrity or availability of a resource. Network monitoring, as a part of intrusion detection, is a common process carried out by network administrators in order to observe the activities in a particular network. As it is a continuous process, the size of data that must be monitored varies with the bandwidth and latency of the network, which can be in several Gigabits per second. This makes the application data-intensive. Furthermore, networks are not restricted to a small room or a building and can spread throughout the globe. In such a distributed setting, it becomes critical to optimize the cost of data transfer from distributed sources in order to perform very frequent network monitoring. The situation becomes more challenging when the raw data, which can be used to detect such attacks are globally distributed. Hence, in this paper, we focus on minimizing the cost of such distributed analysis.

Data mining techniques have become prominent in detecting intrusions [7, 6]. Detecting intrusions can be considered as finding the outliers (or unusual activities) and, hence, data mining can be easily applied to perform this task. We represent this process graphically in Figure 2.

Figure 2 describes the training, testing, and real-time processes labelled in the figure as Block A, Block B and Block C, respectively. The first step for training is to collect

some training data, which can be the contents of IP packets, load on particular servers, logs generated by web servers and application servers or any other detail depending upon the environment. Collected data are then represented in a format that is supported by the data mining tool (in our case it is *.arff* format). The next task in training involves pre-processing the data which includes data normalization, standardization and transformation, adding missing values, etc. Once we have the pre processed data, we select a small set of features which are significant. This helps to improve the performance of the system as well as to improve accuracy. Finally, with the reduced training data (which have only a small number of significant features), we apply different algorithms to train corresponding models. Such algorithms include Naive Bayes Classifier, Decision Trees (J48), Support Vector Machines (SMO) and others.

To evaluate the effectiveness of the trained models, we perform the testing on the test data (which are new and are not used for training) (Block B). We repeat the same steps of data representation, data pre-processing and select the same features as selected during the training phase. We then use the trained model to generate output using the test data. Finally, we can select the best performing model based on the accuracy of classification of individual models.

In our experiments, we selected some well known methods for data mining and intrusion detection such as Naive Bayes, Decision Trees and Support Vector Machines. The advantage of Naive Bayes and Decision Trees is that they are highly efficient and generally result in good classification. Support Vector Machines are high quality systems and have good classification accuracy. We must also remember that our objective in this paper is *not to discover the best model for intrusion detection* rather it is to minimize the cost of data transfer and computation when using data from distributed Cloud storage servers for any analysis of intrusion detection in Clouds.

Once, we have selected a single model which is the most accurate, we deploy the same in the real environment (Block C). In this environment, real-time data from distributed data sources are used and we follow the same steps as in the testing phase except, we use only the best model and operate continuously as more and more data flow in.

3.3 Intrusion detection application workflow

Figure 3 depicts the intrusion detection workflow derived from the intrusion detection process given in Figure 2. As in Figure 2, the workflow has three distinct sections, each consisting of data mining tasks and data dependencies between tasks. Blocks A, B & C represent the training, testing and real-time data analysis phase, respectively. The input data for Block A are the training data, Block B are the testing data and Block C are the real-time data. The output data from each task are represented as arrow labels. At the end

²http://www.sans.org/resources/idfaq/what_isid.php

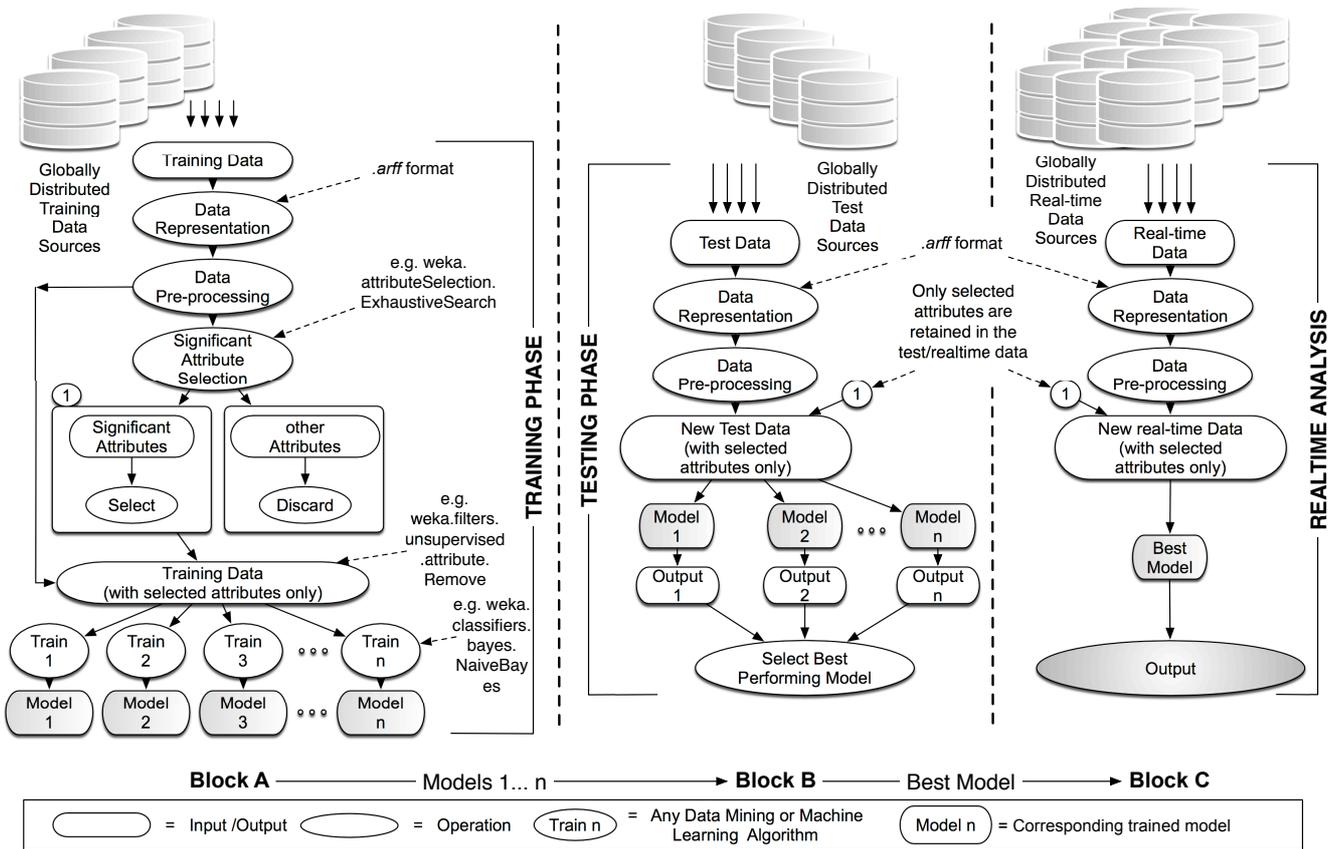


Figure 2: Intrusion Detection Process

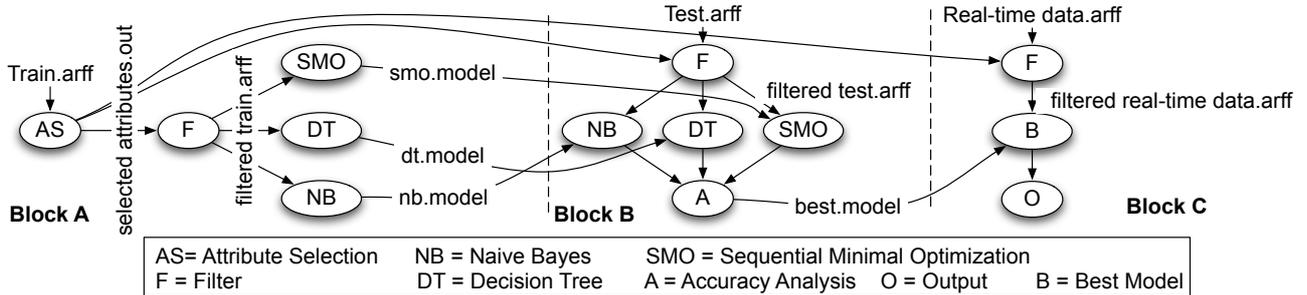


Figure 3: Intrusion Detection workflow

of Block A's execution, three models (nb.model, dt.model & smo.model) files are created, which then becomes input for Block B. Block B's execution generates of the most accurate model (*best.model*) as input for Block C. Block C then applies the model for the real-time data logs to obtain the intrusion classification.

4 Cost Minimization using NLP Model

4.1 Notations and problem

We denote an application workflow using a Directed Acyclic Graph (DAG) by $G=(V, E)$, where $V=\{T_1, \dots, T_n\}$ is the set of tasks, and E represents the data dependencies between these tasks, that is, $tdata^k = (T_j, T_k) \in E$ is the data produced by T_j and consumed by T_k .

We have a set of storage sites $S = \{1, \dots, i\}$, a set of compute sites $P = \{1, \dots, j\}$, and a set of tasks $T = \{1, \dots, k\}$. We assume the 'average' computation time of a task T_k on a compute resource P_j for a certain size of input is known. Then, the cost of computation of a task on a compute host is proportional to the time it takes for computation on that resource. We also assume the cost of unit data access $txcost_{i,j}$ from a storage resource S_i to a compute resource P_j is known. The transfer cost is fixed by the service provider (e.g. Amazon CloudFront) or can be calculated according to the bandwidth between the sites. We assume that these costs are non-negative, symmetric, and satisfy the triangle inequality: that is, $txcost_{i,j} = txcost_{j,i}$ for all $i, j \in N$, and $txcost_{i,j} + txcost_{j,k} \geq txcost_{i,k}$ for

all $i, j, k \in N$. These relations can be expressed as:

$ecost \propto \{\text{execution time or characteristics of resource}\}$
 $txcost \propto \text{bandwidth OR } txcost = (\text{cost of tx of unit data})/\text{site}$
 $\text{total cost of a task } C \leq ecost * etime + txcost * data + \text{overheads}$

The cost-optimization problem is: *Find a feasible set of 'partial' data-sets $\{d_{i,j}^k\}$ that must be transferred from storage host S_i to compute host P_j for each task ($T_k \in V$) such that the total retrieval cost and computation cost of the task on P_j is minimal, for all the tasks in the workflow (not violating dependencies).*

4.2 Non-linear model

Here, we try to get the minimum cost by formulating a non-linear program for the cost-optimization problem, as depicted in Figure 4. The formulation uses two variables y, d and pre-computed values $txcost, ecost, txtime, etime$ as listed below:

- y characterizes where each task is processed. $y_j^k = 1$ iff task T_k is processed on processor P_j .
- d characterizes the amount of data to be transferred to a site. e.g. $d_{i,j}^k = 50.2$ denotes 50.2 units of data are to be transferred from $S_i \Rightarrow P_j$ for task T_k .
- $txcost$ characterizes the cost of data transfer for a link per data unit. e.g. $txcost_{i,j} = 10$ denotes the cost of data transfer from $S_i \Rightarrow P_j$. It is added to the overall cost iff $d_{i,j}^k > 0$ & $y_j^k = 1$.
- $ecost$ characterizes the cost of computation (usage time) of a processor. e.g. $ecost_j = 1$ denotes the cost of using a processor P_j . It is added to the overall cost iff $y_j^k = 1$.
- $txtime$ characterizes the average time for transferring unit data between two sites. e.g. $txtime_{i,j} = 50$ denotes the time for transferring unit data from $S_i \Rightarrow P_j$. It is added to the Execution Time (ET) for every task iff $d_{i,j}^k > 0$ & $y_j^k = 1$.
- $etime$ characterizes the computation time of a task averaged over a set of known and dedicated resources. e.g. $etime_j^k = 20$ denotes the time for executing a task T_k on a processor P_j . It is added to ET iff $y_j^k = 1$.

The constraints can be described as follows:

- (a) & (h) ensure that each task $k \in T$ is computed only once at processor $j \in P$ when the variable $y_j^k > 0$. For partial values of y_j^k , we round up/down to the nearest integer (0 or 1). Tasks are not partitioned or migrated.
- (b) & (c) ensure that partial data transferred and total data required by a task cannot be negative.
- (d), (e), (f) and (g) ensure that cost and time values are all positive.
- (i), (a) & (b) ensure that partial-data are transferred only to the resource where a task is executed. For all such transfers, the sum of data transferred should equal to the data required by the task, which is $tdata^k$.

$$\begin{aligned}
 & \text{Minimize total Cost (C)} \\
 & C = \sum_{i \in S, j \in P, k \in T} d_{i,j}^k * txcost_{i,j} * y_j^k + ecost_j * etime_j^k * y_j^k \\
 & \text{Subject to :} \\
 & (a) \quad \forall k \in T, j \in P \quad y_j^k \geq 0 \\
 & (b) \quad \forall i \in S, j \in P, k \in T \quad d_{i,j}^k \geq 0 \\
 & (c) \quad \forall k \in T \quad tdata^k \geq 0 \\
 & (d) \quad \forall i \in S, j \in P \quad txcost_{i,j} \geq 0 \\
 & (e) \quad \forall i \in S, j \in P \quad txtime_{i,j} \geq 0 \\
 & (f) \quad \forall k \in T, j \in P \quad ecost_j \geq 0 \\
 & (g) \quad \forall k \in T, j \in P \quad etime_j^k \geq 0 \\
 & (h) \quad \sum_{j \in P} y_j^k = 1 \\
 & (i) \quad \sum_{i \in S, j \in P} y_j^k * d_{i,j}^k = tdata^k \\
 & (j) \quad \sum_{i \in S, j \in P, k \in T} y_j^k * d_{i,j}^k = \sum_{k \in T} tdata^k \\
 & \text{Execution time of task } k (ET^k) \\
 & ET^k = \sum_{i \in S, j \in P, k \in T} (d_{i,j}^k * txtime_{i,j} * y_j^k) + etime_j^k * y_j^k
 \end{aligned}$$

Figure 4: NLP-model

- (j) ensures that the total data transfer for all the tasks are bounded by the sum of data required by each task. This is important for the solvers to relate (h), (i) & (j),
- (i) & (j) combined ensure that whenever partial-data $d_{i,j}^k$ is transferred to a compute host P_j , then a compute host must have been selected at j ($y_j^k = 1$), and that total data transfer never exceeds the bound $tdata^k$ for each task and in total.

To get an absolute minimum cost, we map the tasks in the workflow onto resources based only on cost optimization (not time). This eliminates the time dependencies between tasks. However, the task to compute-resource mappings and data-source to compute-resource mappings minimizes the cost of execution *but not the makespan*. The execution time of a task (ET^k) is calculated based on the cost-minimized mappings given by the solver. The total: $\sum_{k \in T} (ET^k + \text{waiting_time})$ is the makespan of the workflow with the minimum cost, where the *waiting_time* denotes the minimum time a task has to wait before its parents finish execution.

5 Cost Minimization for The Intrusion Detection Application

In this section, we describe the method we used to solve the non-linear program we formulated in Section 4. We then describe how we applied the solution for minimizing the total cost of execution to the intrusion detection application

workflow.

NLP-solver: A program was written using the Modelling Language for Mathematical Programming (AMPL) [5]. We used DONLP2[11], a non-linear program solver, to solve our NLP-model.

Partial-data retrieval and task-to-resource mapping:

Based on the integer values of y_j^k given by DONLP2, we statically mapped the tasks in the intrusion detection application workflow to each compute resource P_j . Data retrievals are also fixed for each ready task from each S based on the value of $d_{i,j}^k$ and $y_j^k = 1$. The steps of mapping and data retrievals are given in Algorithm 1. The heuristic computes the values for task mapping y_j^k and $d_{i,j}^k$ for all the tasks in the beginning according to the solution given by a NLP-solver. As all the tasks in the workflow are mapped initially, the *for* loop preserves the dependencies of the tasks by dispatching only the ready tasks to the resources. For dispatched tasks, partial data retrievals to the assigned compute resource occur from chosen resources. All child tasks wait for their parents to complete, after which they appear in the ready list for dispatching. The scheduling cycle completes after all the tasks are dispatched successfully. The output data of each completed task is staged back to the Cloud storage as part of the task’s execution. The Cloud storage should ensure that the files are distributed to the edge-servers within certain time bound such that child tasks do not have to wait for availability of data longer than downloading directly from the Cloud’s central server.

Algorithm 1 Scheduling Heuristic

- 1: Compute y_j^k & $d_{i,j}^k$ for all tasks by solving the NLP
- 2: **repeat**
- 3: Get all the ‘ready’ tasks in the workflow
- 4: **for** each task $t_k \in T_{ready}$ **do**
- 5: Assign t_k to the compute resource P for which $y_j^k = 1$
- 6: Fix partial data transfers $d_{i,j}^k$ from S_i to the compute resource P_j for which $y_j^k = 1$
- 7: **end for**
- 8: Dispatch all the mapped tasks for execution
- 9: Wait for *POLLINGTIME*
- 10: Update the ready task list
- 11: (Upload output files of completed tasks to the storage central for distribution)
- 12: **until** there are unscheduled tasks in the ready list

6 Experimental Setup

In this Section, we describe Intrusion Detection data and tools, the experimental setup and the results.

6.1 Intrusion detection application data and tools

Data: For our experiments, we used part of the benchmark KDD’99 intrusion data set³. We use 10 percent of the total training data and 10 percent of the test data (with corrected labels), which are provided separately. Each record

³<http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>.

Table 1: Distributed Compute Resources

Physical Compute Nodes	cores	Ex cost \$/hr	DTx cost \$/GB	Nearest Region
rotegg.dps.uibk.ac.at	1	\$0.10	\$0.170	Europe
aquila.dacya.ucm.es	1	\$0.10	\$0.170	Europe
tsukuba000.intrigger.omni.hpcc.jp	8	\$0.80	\$0.221	Japan
omii2.crown-grid.org	4	\$0.40	\$0.210	China
snowball.cs.gsu.edu	8	\$0.80	\$0.170	US
node00.cs.binghamton.edu	4	\$0.40	\$0.170	US
belle.csse.unimelb.edu.au	4	\$0.40	\$0.221	Japan
manjra.csse.unimelb.edu.au	4	\$0.40	\$0.221	Japan

in the data set represents a connection between two IP addresses, starting and ending at defined times and protocol. Furthermore, every record is represented by 41 different features. Each record represents a separate connection and is hence considered to be independent of any other record. Training data are either labeled as normal or as one of the 24 different types of attack. These 24 attacks can be grouped into four classes; Probing, Denial of Service (DoS), unauthorized access from a remote machine (R2L) and unauthorized access to root (U2R). Similarly, test data are also labeled as either normal or as one of the attacks belonging to the four attack groups.

The total data used by the intrusion detection workflow (Figure 3) is divided into 30MB, 60MB, 90MB and 120MB. This was achieved by filtering the training, testing and real-time data by random sampling.

Data mining tool: To perform data mining we used algorithms implemented in an open source WEKA library [13]. All data mining was performed using WEKA classes as described in Figure 2. The advantage of using Weka is that it implements a large number of mining algorithms and can be rapidly used to compare different methods.

Classification model: We used three types of probabilistic classification models: Naive Bayes, decision tree and Sequential Minimal Optimization (SMO), from the WEKA library.

Frequency of intrusion detection: The number of log-data analysis for detecting intrusion varies depending on the characteristics of the log data. To reflect all types of scenarios, we perform the real-time log-data analysis for 10 times. We interpolate the cost for 10,000 times execution by multiplying the cost of 10 executions multiplied by 1000.

6.2 Middleware and tools

We used Gridbus-Workflow Engine⁴ [10] for scheduling and managing workflow executions on Cloud resources. We use the scheduling heuristic listed in Algorithm 1. Both, multi-site partial downloads and CloudFront downloads were carried out over HTTP using *JPartialDownloader* tool⁵. HTTP/1.1 range requests allow a client to request portions of a resource.

⁴<http://www.cloudbus.org/papers/Workflow-CCPE2009.pdf>

⁵<http://jpd.sourceforge.net/>

6.3 Distributed compute and storage resources

For selecting the nearest storage location of a file relative to a compute resource, we use the functionality of Amazon CloudFront. CloudFront fetches data to a compute resource from the nearest edge-server. The data transfer cost (per GB) from the edge locations is presented in Table 1. The data transfer cost (DTx cost) from the CloudFront to the execution sites is based on the edge location through which the content is served. We assume the data transfer cost to and from a storage location to be equal in all our experiments. This simplifies the model for the selection of storage sites for partial data retrievals and data upload. For partial data retrievals, all the resources listed in Table 1 also served as storage resources. For our experiments, we ignored the data storage cost on Clouds, which could easily be added to the overall execution cost as a constant (e.g. \$0.150 per GB for the first 50 TB / month of storage used⁶).

We used compute resources from US, Europe and Asia as listed in Table 1. The execution cost (Ex cost) on each CPU is calculated based on the number of cores (cost is similar to Amazon EC2 instances) available.

7 Analysis

We now present results obtained by executing the intrusion detection application workflow using globally distributed resources as listed in Table 1.

7.1 Experiment objectives

We conduct the following two classes of experiments:

1. Measure total cost when using **commercial Cloud as content distribution and publicly available compute resources for execution** ($ecost_j = 0, txcost_{i,j} > 0$).
2. Measure total cost of execution when using **commercial Cloud for content storage, distribution and execution** ($ecost > 0, txcost_{i,j} > 0$)

The first class of experiment measures the cost of data transfer if Cloud resources were used only for data distribution and tasks executed on publicly available compute resources. In this scenario, the compute resources in Table 1 served both as storage (mimicking distributed Cloud storage) and compute resources. We use a solution to our model for determining quantity of partial data transfers from the distributed storage such that the transfer cost is minimized. The tasks are mapped to the compute resources such that the partial transfers have minimum cost.

The second class of experiment measures the cost of executing the application on Cloud, with non-zero data transfer and computation costs. In this scenario, our model gives a solution for minimizing both partial data transfers and computation costs, with tasks mapped to resources accordingly. Here too, the compute-servers in Table 1 serve as distributed Cloud storage and compute resources.

⁶<http://aws.amazon.com/s3/#pricing>

We compare the costs obtained from each of the above experiments against the cost incurred when using data-transfers from nearest (w.r.t. the compute resource where the task is assigned) Cloud storage resource. We measure the total cost incurred for transferring data from nearest location by making compute-resource cost: zero (relating to publicly available resources) and non-zero (relating to commercial Cloud resources), consecutively.

We finally compare the cost savings when using NLP based task+data resource selection against simple Round-Robin (RR) based resource selection.

7.2 Results

The results obtained are an average over 15 executions. The cost values in Figures 5,6 and 8 are for executing a single instance of the intrusion detection workflow. The cost values in Figure 7 are the result of executing the workflow 10,000 times (the cost of 10 executions multiplied by 1000).

Output of intrusion detection: The classification and accuracy presented in the following tables are for a single real-time data set.

Table 2: Accuracy of intrusion detection using SMO

Correctly Classified Instances	407214	99.3115 %
Incorrectly Classified Instances	2823	0.6885 %
Total Number of Instances	410037	

In Table 2, we give the accuracy of the best performing model (SMO). Other methods (Naive Bayes & Decision Trees) had lower accuracy. We see that, using SMO, about 99% of instances are correctly classified. Hence, when this system is deployed in real-time environment, we can expect similar accuracy of classification.

In Table 3, we present the classification using the SMO with the help of a confusion matrix which lists the classification per class (number of correctly and incorrectly classified instances for each class of intrusion). For example, in row 3, we see that 3018 instances are correctly classified as probes while 371 probes are incorrectly classified as normal, 53 probes are incorrectly classified as Denial of Service (DoS), 0 probes are incorrectly classified as unauthorized access from a remote machine (R2L) and 0 probes are incorrectly classified as unauthorized access to root (U2R).

Data in Cloud and execution on public compute resources: Figure 5 compares the cost of transferring data to compute resources between NLP-solver based source selection and single source selection given by CloudFront. We set the execution cost to zero for this analysis. The results show that the cost is minimized when using NLP-solver based storage host selection. As partial data are transferred

Table 3: Classification of data using SMO model

a	b	c	d	e	classified as
80255	43	12	85	4	a = normal
2083	323128	0	0	0	b = DoS
371	53	3018	0	0	c = probe
148	0	0	796	3	d = R2L
10	0	0	11	17	e = U2R

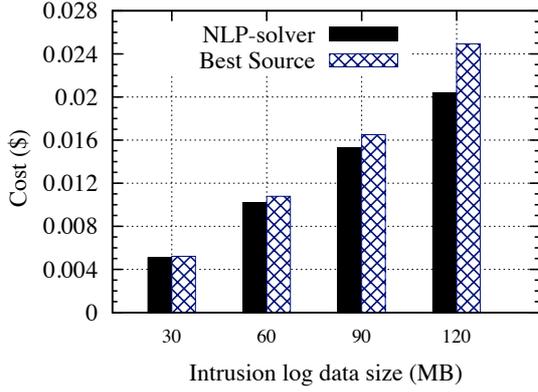


Figure 5: Transfer Cost with no execution cost.

according to the cost of communication, the data transfer cost is divided among all the cheapest links as opposed to the single source. Transferring data using CloudFront becomes more expensive as the size of data increases.

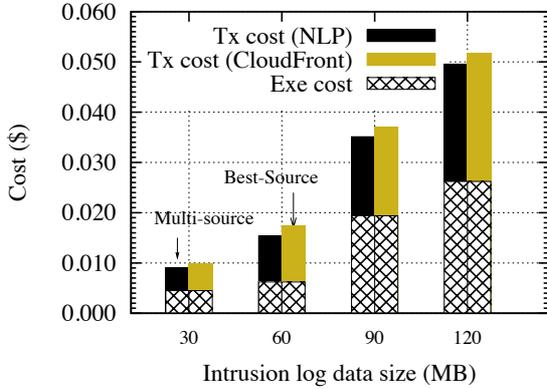


Figure 6: Total cost with computation and transfer costs.

Data and execution on Cloud resources: Figure 6 depicts the total cost of executing the intrusion detection workflow on Cloud resources when using NLP-solver based task-resource mapping and (a) NLP-solver based data source selection (labelled as NLP in the figure), (b) CloudFront based data source selection (labelled as CloudFront in the figure). In this case, the NLP-model relates both the execution costs and data transfer costs into one objective function that is to be minimized. This resulted in the increase of the data-transfer cost when execution cost was minimized and vice-versa. Nevertheless, partial data transfers based on NLP-based data source selection incurred the minimal cost for all range of data sizes.

Total cost savings: Figure 7 depicts the cost of executing the real-time analysis section (Block C) of the intrusion detection workflow (Figure 3) 10,000 times (Block A and B are usually computed only once for a set of data). The cost values for each data group were derived from the cost of 10 executions multiplied by 1000. The most costly approach was when using round-robin based task to resource

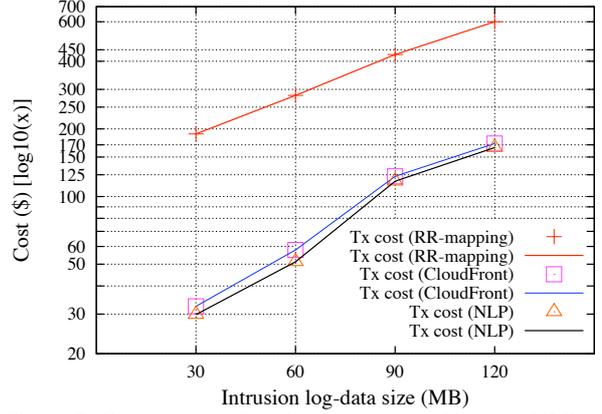


Figure 7: Comparison of total execution cost between NLP based mapping (with and without CloudFront) and round-robin based mapping.

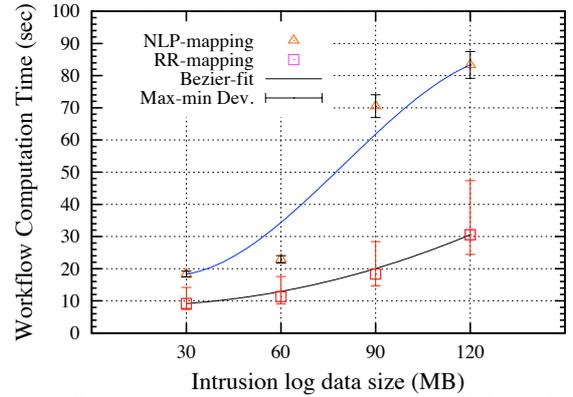


Figure 8: Comparison of execution time between NLP based mapping and Round-Robin based mapping.

mapping algorithm and nearest source data retrievals. This cost was reduced by **77.8% (\$466.8)** when we used the NLP-solver based mapping and multi-source partial data retrieval; and by **76.2% (\$457.2)** when we used NLP-solver based mapping and data retrieval from CloudFront’s best source. This would amount to **savings of three-quarters** of the total expenditure if intrusion detection systems were to be executed on Clouds using our model.

Workflow computation time: We measured the time taken for computing (excluding data transfer time) the workflow under two scenarios:(a) when cost is minimized using our NLP-model, and (b) when time is minimized by a simple round-robin based selection of resources, depicted in Figure 8. When compared against a simple task mapping algorithm such as round-robin, NLP-model based heuristic takes additional time, which increases as the size of the data-set increases as evident from Figure 8. Figure 8 also depicts the maximum and minimum values of execution time for various data sizes. When the compute resource list was randomized, the deviation for the NLP-mapping (mainly due to CPU load) was lower than the RR-mapping (change in type of CPU). The computation time can be reduced by increasing the number of compute resources used,

while still using Clouds for data storage and distribution.

We tabulated the cost of computation and data transfer according to Amazon's current pricing policy in Table 1. The highest computation cost of Amazon Cloud resources is more than the highest data transfer cost⁷. Armbrust et al. [1] have compared the normalized cost of computing resources and WAN bandwidth between 2008 and 2003. Their data clearly shows that the cost/performance improvement is 2.7 times and 16 times for WAN bandwidth and CPU hours, respectively. This trend hints to the fact that data transfer costs are not decreasing as much as computation cost. Hence, for data-intensive applications, total cost savings on communication is a necessity as compared to computation cost.

8 Conclusions

In this work, we presented the execution of an intrusion detection application workflow on Cloud resources, with an objective of minimizing total execution cost. We modeled the cost minimization problem and solved it using a non-linear program solver. Based on the solution, we retrieved data from multiple data sources to the compute resource where a task was mapped. Using our NLP-model we achieved savings of three-quarters of the total cost as compared to using CloudFront's 'best' data source selection, when retrieving data. This cost savings, however, resulted in higher computation time. Computation time could be reduced by using larger number of compute resources. We conclude that by dividing data transfers to distributed datacenters or storage Clouds in proportion to their access cost, users can achieve significant cost savings.

As part of our future work, we would like to look into multi-objective optimization algorithms that balances cost and time for data-intensive workflows on Clouds.

Acknowledgments

This work is partially supported through Australian Research Council (ARC) Discovery Project grant. We thank University of Tokyo, University of Innsbruck, Poznan Supercomputing Center, Universidad Complutense de Madrid, Beihang University, Georgia State University, SUNY Binghamton for providing us compute nodes for our experiments. We would also like to thank James Broberg, Anton Beloglazov and Marco Netto of the University of Melbourne.

References

- [1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A Berkeley view of cloud computing. Technical report, University of California at Berkeley, February 2009.
- [2] J. Broberg, R. Buyya, and Z. Tari. Metacdn: Harnessing 'Storage Clouds' for high performance content delivery. *Journal of Network and Computer Applications*, 32(5):1012–1022, 2009.
- [3] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599–616, 2009.
- [4] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good. The cost of doing science on the cloud: the montage example. In *SC '08: Proc. of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–12, Piscataway, NJ, USA, 2008.
- [5] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press, November 2002.
- [6] K. K. Gupta, B. Nath, and K. Ramamohanarao. Layered Approach using Conditional Random Fields for Intrusion Detection. *IEEE Transactions on Dependable and Secure Computing*. In Press.
- [7] W. Lee, S. J. Stolfo, and K. W. Mok. A data mining framework for building intrusion detection models. In *IEEE Symposium on Security and Privacy*, pages 120–132, 1999.
- [8] R. McClatchey, A. Anjum, H. Stockinger, A. Ali, I. Willers, and M. Thomas. Scheduling in data intensive and network aware (diana) grid environments. *Journal of Grid Computing*, 5(1):43–64, 2007.
- [9] Microsoft. Windows Azure Platform .NET Services. <http://www.microsoft.com/azure/netservices.aspx>.
- [10] S. Pandey, W. Voorsluys, M. Rahman, R. Buyya, J. Dobson, and K. Chiu. A grid workflow environment for brain imaging analysis on distributed systems. *Concurrency and Computation: Practice and Experience*, In Press, 2009.
- [11] P. Spellucci. A sqp method for general nonlinear programs using only equality constrained subproblems. *Mathematical Programming*, 82:413–448, 1993.
- [12] S. Venugopal, R. Buyya, and K. Ramamohanarao. A taxonomy of data grids for distributed data sharing, management, and processing. *ACM Computing Surveys*, 38(1), 2006.
- [13] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco, 2 edition, 2005.
- [14] Y.-S. Wu, B. Foo, Y. Mei, and S. Bagchi. Collaborative intrusion detection system (cids): A framework for accurate and efficient ids. In *Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC 03)*. Purdue University, Applied Computer Security Associates, 2003.
- [15] M. Zeller, R. Grossman, C. Lingenfelder, M. R. Berthold, E. Marcade, R. Pechter, M. Hoskins, W. Thompson, and R. Holada. Open standards and cloud computing: Kdd-2009 panel report. In *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 11–18, NY, USA, 2009. ACM.

⁷assuming transferring 1GB from Amazon takes 1 hour of CPU time