Large Data and Computation in a Hazard Map Workflow Using Hadoop and Multiscale Emulator Methods

E. Ramona Stefanescu*, Shivaswamy Rohit* and Abani K. Patra*
*Department of Mechanical and Aerospace Engineering
University at Buffalo, Buffalo, New York 14260–4400
Email: abani@buffalo.edu

***Abstract*—Three main processes are required in order to construct a simulation based probabilistic hazard map for volcanic debris: a) a good physical model, i.e. a simulator, with error control; b) an approach to uncertainty quantification usually involving an ensemble of simulator runs and an easily sampled surrogate model; c) a multiscale process to analyze the simulator output into a map of the probability that some hazard criteria will be met. The above processes lead to large amount of data and CPU intensive computing. This work advances state of the art in the last area, by deploying a novel programming model; and also by using a hierarchical approach in creating a fast surrogate of the simulator. Handling the huge amount of data and computing is a primary concern in constructing a hazard map on distributed memory platforms. We describe a framework that we built using Hadoop to parallelize the data and compute intensive operations. We make use of Hadoops MapReduce framework to improve the speed of hazard map creation.**

***Keywords*-Uncertainty Quantification; Multiscale Emulator; Hadoop;**

I. Introduction

For the purpose of this paper, we will assume that a suitable physical model of geophysical mass flow is available (TITAN2D). Any simulator will naturally require values for certain input parameters. If the input values for a future event of interest were known exactly, then a hazard map could be generated from a single simulation evaluated at those inputs. Since we lack perfect knowledge of the future, it is necessary to examine flow behavior over a range of inputs. While there are some sampling methods, such as Monte Carlo, that are *nominally independent* of the number of dimensions, there are far too expensive to use to create a feasible hazard map. The solution we came up to this problem can be summarized as:

- Running a *relatively* small number of simulations (a few hundred to a few thousand) followed by
- Constructing a meta-model (a model of the simulator) to act as a fast surrogate for the expensive simulator, and then
- Generating the hazard map from the fast surrogate.

In this work we introduce a multi-resolution scheme for an emulator construction on a high-dimensional parameter space. The proposed scheme overcomes some limitations of the parameter selection in the Bayesian emulator, which always involves repeated inversion of error "correlation matrix", R. The requirement of matrix inversion restricts emulators to small amounts of data mostly because for "large" N: R is poorly conditioned and cost of inverting matrix is $\mathcal{O}(N^3)$ operations. Our scheme is based on mutual distances between data points and on a continuous extension of Gaussian functions. It uses a corse-to-fine hierarchy of the multi-resolution decomposition of a Gaussian kernel. It generates a sequence of approximations at the given function on the data, as well as their extensions to any newly-arrived data point. The subsampling is done by interpolative decomposition of the associated Gaussian kernel matrix in each scale in the hierarchical procedure. In this way a well-conditioned basis is identified and used in the extension/ extrapolation process.

II. Computational challenages

A hazard map, as stated here, is a predictive map for a region which provides a probabilistic measure of a hazard (e.g. geophysical flow reaching certain depths that can be considered hazardous/critical). Simple uncertainty quantification using a Monte Carlo approach for generating such hazard maps will require $O(10^6)$ such simulations – beyond current computational capabilities. Since each flow simulation generates upwards of 2GB of data, a full ensemble with 2048 simulations generates almost 600GB of data which we have to then use to construct emulators. The computational difficulties include managing and accessing select entities from the large data and of processing it using compute intensive operations. Our approach to addressing these difficulties is to decouple the data mining and computationally intensive tasks through carefully crated workflows. We thus, employ the popular open source version of the Map-Reduce model, namely Hadoop, both in conjunction with a high performance cluster. Problems involved in data extraction, movement over the network, and replication are addressed. The proposed computational methodologies also allow us to greatly improve the resolution of the developed hazard maps by using a hierarchical approach (rather than the simpler tessellation based localization used earlier [1]) allowing access to more data in the inference process and hence

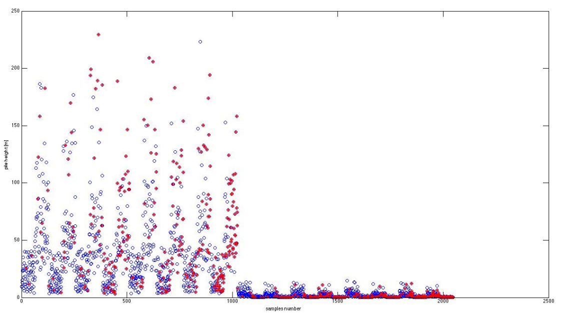

Figure 1. Blue dots represent the maximum pile heigh at a selected location for each LHD sample point, the red starts represent the multiscale data samples.

developing a more accurate localization of the covariance used in the hazard map construction.

III. Multiscale emulator

In the analysis of large scale simulations of complex dynamical systems, where the notion of time evolution comes into play, important problems are the identification of variables that capture the time evolution of the system. Traditionally, creating hazard maps has required detailed knowledge of past events which has been recorded in the volcano's geologic record. Knowledge of this history is frequently incomplete or even entirely unavailable. Our goal is to produce a map showing the probability that each East-North point has of being inundated by a volcanic landslide with a flow-depth greater than a critical threshold from a collection of simulator runs whose parameters are drawn from a distribution that represents a volcanologist's best guess of the range of possible scenarios. We consider a study case for Soufrière Hills, Montserrat, for which ranges of the flow volume, bed friction, internal friction and direction are available. The 4-dimensional input parameter space is sampled using a simple space filling design like Latin Hypercube Design (LHD) to obtain 2048 sets of input. Simulations are performed at each sample point using the TITAN2D model to generate a map of maximum pile height, $h_{max}(\mathbf{x})$ as function of position. If we plot the maximum pile height at a specific location in space for each sample point we can easily observe the binning introduced by the LHD sampling (Fig. 1 - blue dots). In LHD, each random direction (random variable or input) is divide into N_{bin} bins of equal probability and one random value is selected in each bin [2]. By performing a multiscale data sampling we identify a well-conditioned basis of a low rank Gaussian kernel matrix, used in identifying scattered data sampling (Fig. 1 - red stars). Using the obtained sample data, h_{max} is evaluated at re-sample points.

Typical usage involves evaluating the fast surrogate at hundreds of thousands or millions of re-sample input points. The number of samples needed to generate the fast surrogate are exponential in the number of dimensions. This effectively limits its application to when there are three or fewer uncertain dimensions. This implies that we need to change the representation of the 2048 data sets, into a low-dimensional that describe the data in a faithful manner. In order to achieve this goal, we use a technique that relies on graph-based algorithms. Weighted graphs are employed to represent the "geometry" based on the local similarity or interaction between the data points. Since each of the data sample is represented by a collection of numerical attributes, the condition of two nodes to be connected is based on the proximity of the corresponding data points in the feature space.

A graph $G = (V, E)$ is characterized by a set of vertices $V = \{1, \ldots, m\}$ and a set of edges $E = \{e_{ij} | i, j \in V\}$. Let $A = [a_{ij}]$ be the $m \times m$ adjacency matrix such that a_{ij} represents the weight of edge e_{ij}. If there is no edge between vertices i and j then $a_{ij} = 0$. Clustering the vertices V into c disjoints sets $V_1, \ldots, V_c$ with $m_i = |V_i|$, the adjacency matrix will have the following form:

$$A_{m,n} = \begin{bmatrix} A_{11} & \cdots & A_{1c} \\ A_{2,1} & \cdots & A_{2,n} \\ \vdots & \ddots & \vdots \\ A_{c1} & \cdots & A_{c,c} \end{bmatrix} \tag{1}$$

where each diagonal block $A_{ii}, i = 1, \ldots, c$, is an $m_i \times m_i$ matrix that can be considered as a local adjacency matrix for cluster i. The off-diagonal $m_i \times m_j$ blocks A_{ij} with $i \neq j$, contain the set of edges between vertices belonging to different clusters.

In a perfectly clusterable graphs, the off-diagonal blocks will not contain any edges, thus yielding $A_{ij} = 0$, and the graph will consist of c disconnected components. In a realistic scenario, with a graph forming good clusters most of the edges will be contained within the diagonal blocks A_{ii}, while the off-diagonal blocks A_{ij} will contain only a few edges. The decay in the A's spectrum is a measure of the connectivity of the points in the graph.

- One extreme case corresponds to the graph where all the all the nodes are disconnected. This leads to A being equal to the identity operator and thus to a flat spectrum.
- Another case is the graph where all the nodes are connected to all the other nodes weights equal to 1. In this case, A has one eigenvalue equal to 1, and all other eigenvalues are equal to 0 (we obtain the fastest decay possible for the a diffusion operator).
- Usually the spectrum of the heat kernel decays smoothly.

Many dimensionality reduction methods involve a spectral decomposition of large matrices, whose dimensions are proportional to the size of the data, has high computational cost. The n observations $f_1, f_2, \ldots, f_3$ are considered the data points. When the covariance of the data points is unknown, an artificial function has to be chose. A Gaussian covariance is a popular choice:

$$g_\epsilon(x, x') = exp(- \parallel x - x' \parallel^2 / \epsilon) \quad (2)$$

where $\parallel \cdots \parallel$ constitutes a metric on the space (Euclidean distance in our case). The corresponding covariance (affinities) is

$$(G_\epsilon) = g_\epsilon(x_i, x_j),\ i, j = 1, 2, \ldots, n. \quad (3)$$

The combination of clustering and low rank approximation gives a better approximation of the original graph. A standard low rank computation is likely to only extract information from the largest or a few dominant clusters. The $Nyström$ method, vastly used for out-of-sample extension has three significant disadvantages: (a) Diagonalization of G costs $\mathcal{O}(n^3)$ operations; (b) G may be ill-conditioned due to fast decay of its spectrum, and (c) it is unclear how to choose the length parameter ϵ since the output is sensitive to the choice of ϵ. To overcome these limitations a multiscale approach is used: a sequence of Gaussian kernel matrices $G_s, s = 0, 1, \ldots,$ whose entries are $(G_\epsilon) = g_\epsilon(x_i, x_j)$, where ϵ_s is a positive monotonic decreasing function of s, which tends to zero as the scale parameter s tends to infinity (i.e. $\epsilon_s = 2^{-s}$, $s = 0, 1, \ldots.$).

By the application of a randomized interpolative decomposition(ID) to G_s, a well-conditioned basis is identified for it numerical range. In each scale f is decomposed into a sum of its projections on this basis and it is extended as $\bar{f}_* = G_* G^{-1} f$. In addition, selection of the proper columns in G_s is equivalent to data sampling of the associated data points.

The method requires no grid. It automatically generates a sequence of adaptive grids according to the data distribution. It is based on the mutual distances between the data points and on a continuous extension of Gaussian functions. In addition, most of the costly computations are done just once during the process, independently of the number of the extended data points since they depend only in the data and on the given function.

IV. Hazard Map Generation – Data Management and Computing

The simulator naturally require values for certain input parameters, such as a digital model of the terrain, the volume of the flow, material properties, and other initial conditions (starting location of the flow and a preferred initial direction). To understand the complexity of the problem we present an outline of steps involved in a usual process of Hazard map generation.

- **Step 0**: The first step is to run the simulator at well chosen inputs. The input parameters are sampled using a simple space filling design like Latin Hypercube to obtain 2048 sets of input. Multiprocessor Titan2D simulations of these inputs and post processing results in 2 gigabytes(GB) of flowdata per sample in the form of flow height records.
- **Step 1**: The construction of the hazard map requires us to sample a tensor product of the input parameters and 2 space dimensions which results in as many as 10^8 data points. Emulator construction on this very large set is unaffordable so a simple contouring and decimation strategy is used to create a smaller set on which we construct the emulator. This downsampling is introduced to reduce their number to the order of 10^6. Furthermore, resamples from the generated emulator surface (for the final probability calculation) are also required to be generated and can be as many as 10^{10} in number.
- **Step 2**: The size of the downsized data set makes it computationally impossible to fit a single emulator using all the data at once, which warrants the need for piecewise emulator obtained by localizing the covariance. The neighbor search used in identifying the regions for localization is thus an important prerequisite for the functioning of the emulator and requires both sample and resamples to be searched from among the samples for neighbors. Both neighbor search and downsampling are highly data intensive tasks which require little computation but scanning of large datasets.
- **Step 3**: Using neighborhood data, emulator is constructed about the sample points through an iterative process. The functioning of emulator can be understood from the following equations:
  g(y) being the matrix of basis functions evaluated at the resample points and β being the vector of least square co-efficients. R is the matrix of the correlation functions at x such that R$_{i,j}$ = r$_i$(x$_j$) = r$_j$(y$_i$) and σ is the variance.
  $s(x) = \beta G(x) + \hat{\epsilon}$ is the response function.
  ϵ = s(x) - G(x)β the true error evaluated at the sample points. θ_n is the vector of hyper-parameters or roughness parameters and Ndim is the number of dimensions

Algorithm 1: Initial setup

```
Data: A data set $D = \{x_1, \ldots, x_n\} \in \mathcal{R}^d$ (based on
      LHS design)
Result: A function $f = [f_1, \ldots, f_n]^T$
while $i \leq n$ do
    Run TITAN2D simulation for each $x_i$.
    Use a downsample algorithm $\rightarrow N$ downsample
    points.
    For each $N$ generate an $f$ based on maximum
    height and apply the response surface calculation
    algorithm.
    Perform multiscale d
end
```

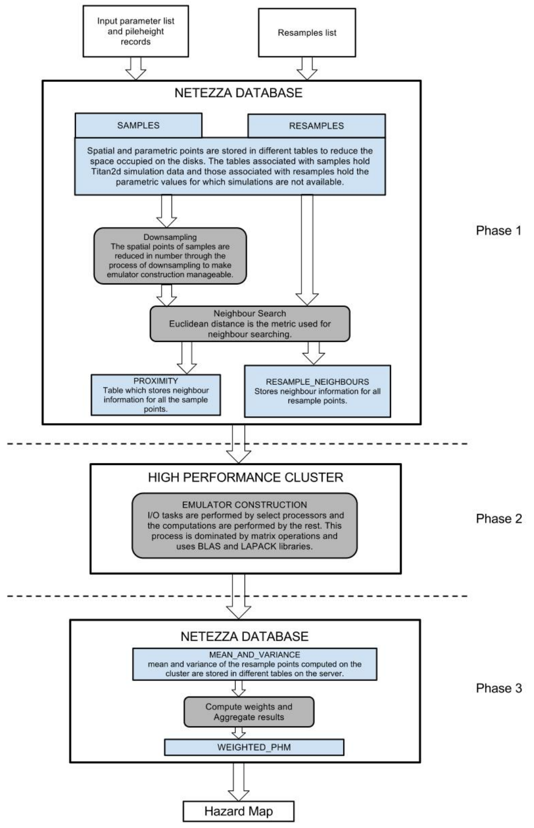
Input parameter list and pileheight records
Resamples list
NETEZZA DATABASE
SAMPLES
RESAMPLES
Spatial and parametric points are stored in different tables to reduce the space occupied on the disks. The tables associated with samples hold Titan2d simulation data and those associated with resamples hold the parametric values for which simulations are not available.
Downsampling
The spatial points of samples are reduced in number through the process of downsampling to make emulator construction manageable.
Neighbour Search
Euclidean distance is the metric used for neighbour searching.
PROXIMITY
Table which stores neighbour information for all the sample points.
RESAMPLE_NEIGHBOURS
Stores neighbour information for all resample points.
Phase 1
HIGH PERFORMANCE CLUSTER
EMULATOR CONSTRUCTION
I/O tasks are performed by select processors and the computations are performed by the rest. This process is dominated by matrix operations and uses BLAS and LAPACK libraries.
Phase 2
NETEZZA DATABASE
MEAN_AND_VARIANCE
mean and variance of the resample points computed on the cluster are stored in different tables on the server.
Compute weights and Aggregate results
WEIGHTED_PHM
Phase 3
Hazard Map

Figure 2. Illustration of the integrated workflow using Netezza architecture.

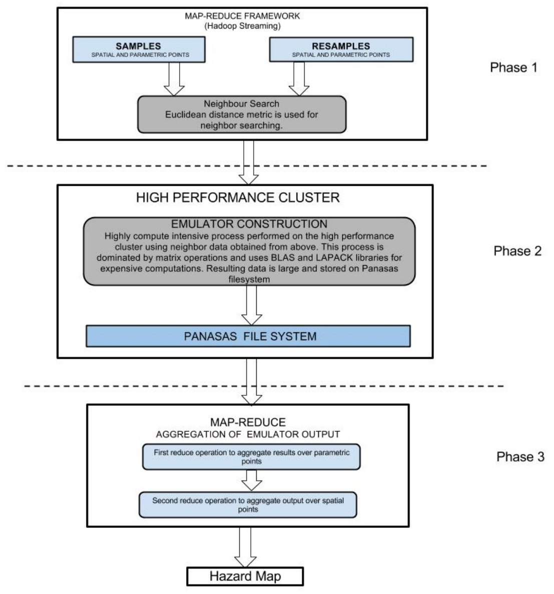
MAP-REDUCE FRAMEWORK
(Hadoop Streaming)
SAMPLES
SPATIAL AND PARAMETRIC POINTS
RESAMPLES
SPATIAL AND PARAMETRIC POINTS
Neighbour Search
Euclidean distance metric is used for neighbor searching.
Phase 1
HIGH PERFORMANCE CLUSTER
EMULATOR CONSTRUCTION
Highly compute intensive process performed on the high performance cluster using neighbor data obtained from above. This process is dominated by matrix operations and uses BLAS and LAPACK libraries for expensive computations. Resulting data is large and stored on Panasas filesystem
PANASAS FILE SYSTEM
Phase 2
MAP-REDUCE
AGGREGATION OF EMULATOR OUTPUT
First reduce operation to aggregate results over parametric points
Second reduce operation to aggregate output over spatial points
Phase 3
Hazard Map

Figure 3. Illustration of the integrated workflow with Map-Reduce framework

Algorithm 2: Response surface calculation

```
Data: A data set $D = \{x_1, \dots, x_n\} \in \mathcal{R}^d$, $T > 0$, a
      new data point $x_* \in \mathcal{R}^d$, a function
      $f = [f_1, \dots, f_n]^T$ to be extended and an error
      parameter $err \geq 0$.
Result: An approximation $F = [F_1, \dots, F_n]^T$ of $f$ on
      $D$ and its extension $F_*$ to $x_*$
Set the scale parameter $s = 0$, $F^{(-1)} = 0 \in \mathcal{R}^n$ and
$F_*^{(-1)} = 0$.
while $\| f - F^{(-1)} \| > err$ do
    Form the Gaussian kernel $G^{(s)}$on $D$ with $\epsilon_s = \frac{T}{2^s}$;
    Estimate numerical rank $l^{(s)}$ of $G^{(s)}$;
    Generate $A$ whose entries are i.i.d Gaussian
    random variables of zero mean and unit variance:
    $W = AG^{(s)}$;
    Apply pivoted QR on $W \rightarrow WP_R = QR$;
    Split R and Q st.
        $\left[\begin{array}{c|c} R_{11} & R_{12} \\ \hline 0 & R_{22} \end{array}\right]$
    and [ Q1 | Q2] ;
    $S = Q_1 R_{11}$ ;
    Columns of $S$ constitute a subset of of the columns
    of $W$. The corresponding columns of $G^{(s)}$ are
    collected into a matrix $B$, so that the column of $j$th
    column of $B^{(s)}$ is the $i_j$th column of $A$. The
    sampled dataset is $D_s$.;
    Calculate the pseudo-inverse $(B^{(s)})^\dagger$ of $B^{(s)}$.;
    Calculate the coordinates vector of the orthogonal
    projection of $f^{(s)}$ on the range of $B^{(s)}$ in the basis
    of $B^{(s)}$'s columns $c = (B^{(s)})^\dagger f$;
    Calculate the orthogonal projection of $f$ on the
    columns of $B^{(s)}$, $f^{(s)} = B^{(s)} c$;
    Form the matrix
    $G_*^{(s)} = [g_\epsilon(x_*, x_{s_1}) \dots g_\epsilon(x_*, x_{s_{l^{(s)}}})]$;
    Calculate the extension $f_*^{(s)} = G_* c$;
    Set $F^{(s)} = F^{(s-1)} + f^{(s)}$, $F_*^{(s-1)} + f_*^{(s)}$, s=s+1;
end
```

associated with the data set.
At each iteration β and R^{-1} are computed using updated values of hyper-parameters (θ). Mean and variance are then evaluated for the resamples and adjusted using bayes linear equations. Typically, a hazard map requires constructing a few million emulators. The emulator construction dominated by O(n^3) matrix operations is a highly compute intensive process but also embarrassingly parallel.

- **Step 4**: In the last stage of hazard map construction, emulator output is aggregated using barycentric weights. This involves scanning the dataset, computing the euclidean distance of the samples that influence a resample point, evaluating their weights and assembling the results.

V. Integrated Workflow for data and compute intensive tasks

There are three dominant phases of Hazard Map generation namely: 1) Downsampling and neighbor searching, 2) Emulator construction, and, 3)Aggregation of resulting data. Our computational model based on a *divide and conquer* strategy, segregates these phases and performs them on either Netezza server or the high performance cluster depending on the computational requirements.

A. Downsampling and neighbor search

Both downsampling and neighbor search are ideally suited for distributed systems. For multidimensional dataset, such as ours, operations like neighbor search are afflicted with the *curse of dimensionality*. Several tree and clustering based methods have been proposed but most converge to sequential search for higher dimensions and/or are difficult to implement. Neighbor search operation on a distributed environment like Netezza server or on a cluster through Map-Reduce implementation like Hadoop allows for a much simpler algorithm and adapts to large datasets. Downsampling and Neighbor search on the Netezza system were easily accomplished because of its massively parallel architecture. Netezza's high performance stems from filtering data close to disk and from its MPP architecture. Since a significant amount of data movement is curtailed through the use of FPGA filtering, we abstained from using complex algorithms in favor of brute force techniques. All Netezza based implementations were in plain SQL which ensured parallelism and high performance.

We also tested same operations on the high performance cluster using python scripts, relying on Hadoop Streaming API for task distribution and scheduling. In a distributed environment the underlying algorithm for neighbor search remains essentially the same as the one used on Netezza server. The mapper computes the distances between two sets of data (X and Y) and prints out the result as key-value pair, key being the indices of dataset X and value being the indices of dataset Y and the euclidean distance. In the absence of a customized partitioner same keys are guaranteed to be dispatched to the same reduce task. The reduce operation merely involves printing out the indices of set Y against the keys obtained from set X. Hadoop based implementation as in the case with Netezza was simple and concise.

In both our implementations euclidean distance was the metric for neighbor search. The parametric neighbor search was performed independent of spatial neighbor search by separately treating the two sets of dimensions. Though the brute force method of neighbor search has a complexity of O(n^2), it is well suited for distributed environment owing to good scalability. the Netezza architecture is ideally suited for such methods. The distance based neighbor search also made it possible to more precisely define the region of neighborhood which could not have been achieved using tessellation based neighbor search. A notable advantage of neighbor search on the database is that it can be easily extended to even higher dimensions without the need of any significant changes in the implementation.

Database operations played a pivotal role in our computing model by making the cumbersome task of data-mining on the cluster a rather elegant and transparent process. Netezza's datawarehousing capabilities ensured that all data could be stored at one place as opposed to having it scattered across a large number of flat files on the cluster. Using SQL, clean and concise codes could be written with provision for easily making changes. Several hundreds of lines of MATLAB code were replaced with less than hundred SQL queries. Data warehousing capability and "under the hood" data movement offered an enormous respite from the MATLAB implementation on the cluster where data movement was achieved by reading and writing of a large number ASCII files. Table **??** lists some of the tables that were generated on Netezza for one of the cases prior to emulator construction. Tables with rows of the order of millions and even close to 2 billion in case of table $PROXIMITY$ were generated with ease.

B. Emulator Construction

Unlike the data mining operations up to this point, the emulator construction is a computationally expensive process dominated by matrix operations. Though emulator conctruction is an embarrassingly parallel process and Netezza server boasts of massively parallel architecture, it was, for variety of reasons as we describe below, appropriate to siphon data off the database to an architecture that could meet the emulator's computational demands. First, complex algorithms can not be easily translated into a declarative language like SQL. Second, the number of blades available in our systems put a limit on the available processors. Also the explicit use of high performance libraries like *blas* and *lapack* was not possible. Another subtle aspect about emulator which advocates against its construction on the database is that it requires small chunks of data, more precisely the neighborhood data. Netezza is more appropriate for scanning large datasets. We present here the steps taken and the challenges faced in integrating server to the cluster.

The high performance cluster and the Netezza system were connected using Netezza's ODBC driver. Data transfer across the two frameworks was possible largely through the use of named pipes. On linux systems a named pipe is a type of file - FIFO file (FIFO being the acronym for First In

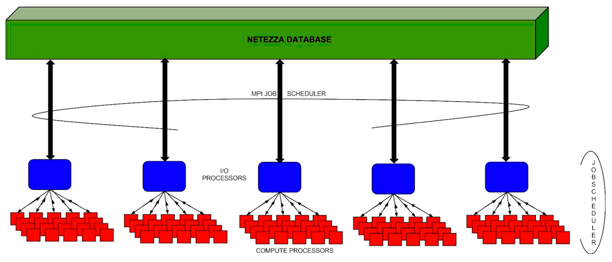

Figure 4. Schematic representation of integration of Netezza with high performance cluster

First Out). A named pipe does not store or buffer data and hence does not occupy storage space on the filesystem.
Loading data from cluster to Netezza was achieved through the *nzload* feature. The *nzload* command is a SQL CLI client application which in conjunction with named pipes that makes it possible to load data to Netezza server from any remote client[3].

Emulator construction requires that each processor request the server for its share of data. Large number of concurrent I/O requests by nodes on the grid can result in I/O bottlenecks and processors starving for data. Such problems have been addressed in the past and techniques such as collective I/O[4], [5] and I/O forwarding[6] have been proposed. I/O forwarding mitigates this problem by passing the I/O requests of the all the nodes to only a select number of nodes called as I/O nodes. We adopted this method for our computing model with some modifications. A few processors on the high performance cluster are identified as the I/O processors and the rest as compute processors. All the processors are assigned a group, with each group having one I/O processor and several compute processors. Each group operates independent of the other with I/O processors responsible for performing all I/O operations on behalf of the compute processors of their respective groups. I/O processors alone communicate with the Netezza server and any transfer of data occurs through named pipes. Multiple small requests are avoided and data is invariably transferred in bulk. I/O processors extract data from Netezza, disseminate it across the compute processors and deliver the data gathered from the compute processors back to the Netezza database. I/O processors of every group draw the neighborhood data of all the downsampled points corresponding to the pileheight simulation number assigned to them from sever, and store it in their data structures. Each compute processor based on the data received from the I/O processor of its group builds an emulator. The mean and variance data of the resample points, computed at the end is sent over the network to I/O processor.
The operations on the cluster are parallelized using MPI (Message Passing Interface). Data is transferred between the I/O and compute processors over the network using MPI protocols. An MPI job scheduler allows compute processors to notify the I/O processors about the status of their completion. When a compute processor finishes constructing emulator about a sample point, it prompts the I/O processor. The I/O processor responds by sending neighborhood data for the next point. The compute processors do not communicate among themselves and are self sufficient with the data received from the I/O processor. The mean and variance information received from the compute processors is allowed to accumulate with I/O processors and dispatched to the Netezza server at regular intervals. Another layer of MPI job-scheduler is responsible for co-ordination between the various groups of processors. A lone processor which holds the metadata maintains communication with the groups and assigns each group a simulation number to work on. Additionally it also prompts Netezza database to "generate statistics" after each load session.

Though we succeeded in integrating server with the cluster and in transferring data over the network, this implementation has apparent shortcomings. Firstly offloading compute intensive tasks from server to cluster defeats the objective of minimizing large data movement. Secondly only a small number of ports can be kept working between the two, to transfer data. Thus, though Netezza can easily house much more data, extracting it to more nodes on cluster is not feasible. It is thus clear that compute intensive tasks like emulator construction expose the vulnerabilities of even a specialized hardware like Netezza database. For the above mentioned reasons and also because of the complexity of implementation, we attempted to test the above model with Hadoop in place of Netezza. Hadoop provides a convenient alternative because it does not require data to be moved into or out of the cluster. The downsampled points stored on files on the Hadoop distributed file system were used as input to the mapper. The mapper itself was simply a python wrapper around the existing code to compute the emulator with output in terms of key value pairs. Hadoop is designed to take in large amount of input and distribute the tasks by spawning mappers on each of the slave nodes. In our application at less than 2GB the input data was very insignificant in size but the generated output was expected to be large. Also the emulator construction is highly cpu intensive. We observed that no more than 2 map tasks could be spawned on each node regardless of the number of cores on them. On certain nodes, with say 12 cores, having only 2 map tasks running was a severe under utilization of the resources. Also, previous experience dictates that at least 500 processors be used for emulator construction to finish the hazard map in reasonable time. Under these circumstances we found it to be more appropriate to perform emulator construction independent of Hadoop environment. The resulting output was stored as key-value pairs on ASCII files.

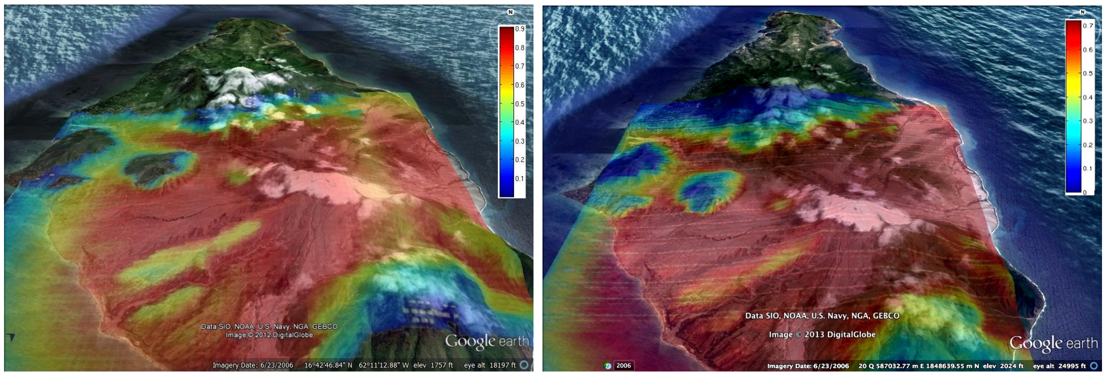
Data SIO, NOAA, U.S. Navy, NGA, GEBCO
Image © 2012 DigitalGlobe
Google earth
Data SIO, NOAA, U.S. Navy, NGA, GEBCO
Image © 2013 DigitalGlobe
Google earth

Figure 5. Figure shows the probabilistic hazard maps for the island of Montserrat in the Caribbean for flow exceeding 0.2m of depth. One on left was generated using Netezza database based model and the one on right using Hadoop API

C. Aggregation Of Results

The first two moments are to be computed for every resample point, which as mentioned earlier, could be as many as 10^{10} in number. Furthermore most of these points occur in the neighborhood of more than one sample point. In the previous work [1], the weights were pre-computed owing to the fixed number of neighbors which made immediate aggregation of data possible. A distance based method of neighbor search is computationally more expensive. The

Netezza database offers aggregate functions and can easily house massive data. The "group by" feature of any database is primarily aimed at operations like reduction and aggregation. Besides, computing the weights required repeated scanning of large parts of the data sets. Netezza is well suited for such aggregation of data. The mean and variance computed by the compute processors is directed to the Netezza server through the I/O processors. The cluster and the Netezza server are connected by 10Gb network at the Center for Computational Research, Buffalo. Using Netezza's nzload command and with multiple named pipes, data is concurrently moved from I/O processors to the respective tables on Netezza. The mean and variance data is massive and runs into billions of rows on the server. It is important that the tables storing such data are already distributed on columns on which they are aggregated. This ensures prior partitioning of data about those columns which significantly reduces the computation time. During the implementation of certain aggregation queries we found, that by having the table (with approximately 2 billion rows) distributed on the columns to be aggregated on, the total time of operation was reduced from approximately 30 minutes to under 3 minutes.

A Hadoop based aggregation of the emulator output was performed by chaining of map-reduce operations. As the term suggests, aggregation, did not require any specific map operation. As the output was required to be aggregated over multiple set of keys, aggregation was split into two separate reduce operations. Both the tasks involved only stdin and stdout operation in the mapper. The reducer was responsible for assembling the weighted mean for records with same keys. The replication factor was fixed at 1 for Hadoop implementations.

The resulting 2048 pileheight records were merged using python scripts and loaded on the Netezza server for downsampling and neighbor searching. The details of some of the tables generated are already provided in Table 1. The dimensions associated with the random variables being vastly different and having very different scales, were all mapped to the same scale of 0 to 1. Matrix operations being expensive, it was essential to keep their sizes small. Thus a limit on the number of neighbors had to be enforced in addition to a limit on the euclidean distance.

The computational details of the hazard map shown in figure 5are enumerated below.

1) The Hazard map was generated using the our computational model comprising of the Netezza server and 504 processors on the high performance cluster.
2) Neighbor search, downsampling and few other operations were performed on Netezza in 30 minutes. The radius of neighborhood was fixed at 0.3 for parametric dimensions (scaled) and 200 metres for spatial dimensions. Furthermore, each sample point could have at most 500 neighbors and this information was stored in the table $PROXIMITY$.
3) 504 processors functioning as 18 independent groups were responsible for emulator construction and the evaluation of mean and variance using Bayes Linear model. Each group had 1 I/O processor and at most 27 compute processors.
4) During 8 hours 47 minutes of wall time on the cluster, approximately 4 million covariance matrices of size 300x300 were computed through an iterative process and mean and variance was computed for approximately 10^{10} resample points.

5) In that time more than 1.5 TB of data was transferred over the network to Netezza server over 18 different pipes. This data was stored on 18 different tables occupying a little more than a total of 50 billion rows.
6) The final step which required scanning billions or rows of data to evaluate the barycentric weights and aggregate the results was also performed on the Netezza server and took 2.5 hours of time.
7) The entire operation completed in little under 12 hours of time.

VI. Results

We put both computing models was put to test with the task of creating the hazard map for the volcano on Montserrat island. 2048 sets of input parameters were generated using Latin Hypercube sampling and Titan2D simulations of these inputs were performed. The probabilistic hazard map shown on the left in figure 7 was generated, in 6 hours time using Netezza based workflow. Downsampling and neighbor search operations were performed in under 30 minutes of time. The computation time on the cluster was reduced to a little more than 2.5 hours using 512 processors on 43 nodes of 12 cores each and by keeping 16 connections open between Netezza and the cluster. The 512 processors were divided into 16 groups with each group having 1 I/O processor and at most 32 compute processors. A maximum limit of 120 was enforced on the size of the covariance matrices. Also the radius of search was reduced from 100 metres for spatial dimensions and from 0.2 for parametric dimensions(scaled). The final aggregation was completed in 2.5 hours, and was performed entirely on Netezza server.

The probabilistic map on the right in figure 5 was generated with Hadoop based model. The neighbor search operations were completed in 20 minutes of wall time. Emulator construction was performed individually by separately dispatching tasks on the cluster. This was completed in approximately 5 hours of time and resulted in 800GB of output data. Hadoop was extensively used for aggregation of results through two separate reduce processes. 200 reduce tasks were spawned over 30 nodes (1 master and 29 slave nodes) for the first reduce operation and the computations ended successfully in approximately 8 hours of wall time reducing 800GB of data to 190GB of output. The second reduce operation was completed with 100 reduce tasks spread over on 20 nodes (1 master and 19 slave) in 1.5 hours of time.

A. Performance Optimization

The Hazard map in figure **??**(b) was generated in 12 hours of wall time. A significant portion of this time was spent on the cluster and in moving the data across, posing a problem which attracted our attention. To understand the reason for long computation time, it must be noted that every successful load session must be followed by the ”generate statistics” command on the table which is loaded. A load session is said to be complete when the pipe that is opened for writing is closed. A single write session on the other hand involves writing all the data accumulated with I/O processor (for sending over to the server), to the open pipe, clearing space in the memory for more data. We were able to improve the performance and reduce the time of computation on the cluster by making communications between the I/O processors and the server less frequent and by introducing minor changes to the design.

More write sessions per load session: Having more write sessions and few load sessions reduced the number requests from the I/O processors to Netezza database to “generate statistics” on the tables resulting in less overall latency.

Single processor handles “generate statistics”: The I/O processors were relieved of the job of prompting the server to ”generate statistics” on the tables, which was delegated to the single processor that also assigns the sample numbers to the I/O processors. This was possible because the load sessions were now less frequent. Fewer interactions between I/O processors and Netezza server, also led to I/O processors dispatching jobs to the compute processors with less interruptions.

Avoid simultaneous aggregation and loading: Simultaneous loading and aggregation of data on the server was to found to impede the performance. By avoiding this and instead scheduling the aggregation process for the end we were able to speed up the process.

Column distribution: Prior distribution of tables based on “columns” of interest ensures appropriate data slicing at the time of loading significantly reducing query response time.

Writing in blocks: For I/O intensive operations such as ours writing in blocks led to notable reduction in time. For our purpose we found 1KB sized blocks to produce the best results.

Implementing these features resulted in cutting the total time taken by over 50%.

VII. Conclusion

Through this paper is we aim to address the simultaneous computational and data challenges in a Uncertainty quantification process through two different approaches - one hardware based and other using a more popular software tool on the traditional cluster. We successfully architected and implemented two functional workflows for the application of generating hazard maps for geophysical flows. Netezza based workflow offered a faster implementation for a moderate sized data such as ours in comparison to Hadoop based workflow. Data mining tasks required minimal work and its ability to perform fast analytics provided quick insight about the data. On the other hand, Netezza is an expensive hardware, with its SPUs (Snippet Processing Units) prone to wearing out. Integrating Netezza with the cluster presented numerous hurdles and the resulting implementation was not fault tolerant. Furthermore, restriction on the number of 'working' ports on Netezza makes the workflow less scalable. Hadoop in contrast is a much cheaper alternative, offering reliable and robust job scheduler and fault tolerance. It made our implementation of the workflow programatically easy and obviated the need to move data from the cluster. It is also more easily scalable. However, modeling, a compute intensive task, such as emulator construction, as a mapper posed a severe problem. Using Hadoop API for tasks deficient in input data but requiring more cpu cycles clearly suggested that it can't be used as an alternative job scheduler for compute intensive tasks.

A number of fundamental changes in the design of hazard map generation were introduced and were largely possible through the use of Netezza server.

With Netezza's data mining capability, tessellation based covariance localization [1], [7] was completely replaced with a more reliable distance based search for neighbors. This not only leads to a more precise control of the region of neighborhood but also makes it more feasible to extend the problem to higher dimensions. In addition search distance for each dimension can be separately incorporated making this model even more versatile. The tessellation based method is firmly dependent on the manner in which spatial points are downsized making $downsampling$ a cornerstone in the process emulator construction. The workflow proposed in this paper attributes less importance to $downsampling$ and offers more flexibility than the previous approach.

It was also comparatively easier to work on a combination of Netezza server and the high performance cluster. The codes were more concise and amenable to changes. Most importantly, information was communicated in a more elegant fashion, with very little need for file handling. Data was pre-dominantly transferred over the network obviating the need to read and write large number of ASCII files. Finally, this model is easily extended to a wide range of problems with the similar challenges of massive data movement and complex computations.

VIII. Conclusion

The conclusion goes here. this is more of the conclusion

Acknowledgment

The authors would like to thank... more thanks here

References

[1] K. Dalbey, *Predictive simulation and model based hazard maps of geophysical mass flows*. State University of New York at Buffalo, 2009.

[2] M. D. McKay, R. J. Beckman, and W. J. Conover, "Comparison of three methods for selecting values of input variables in the analysis of output from a computer code," *Technometrics*, vol. 21, no. 2, pp. 239–245, 1979.

[3] "Netezza data loading guide," Netezza Corporation, October 2010.

[4] R. Oldfield, L. Ward, R. Riesen, A. Maccabe, P. Widener, and T. Kordenbrock, "Lightweight i/o for scientific applications," in *Cluster Computing, 2006 IEEE International Conference on*. IEEE, 2006, pp. 1–11.

[5] D. Kotz, "Disk-directed i/o for mimd multiprocessors," *ACM Transactions on Computer Systems (TOCS)*, vol. 15, no. 1, pp. 41–74, 1997.

[6] N. Ali, P. Carns, K. Iskra, D. Kimpe, S. Lang, R. Latham, R. Ross, L. Ward, and P. Sadayappan, "Scalable i/o forwarding framework for high-performance computing systems," in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*. IEEE, 2009, pp. 1–10.

[7] E. R. Stefanescu, M. Bursik, G. Cordoba, K. Dalbey, M. D. Jones, A. K. Patra, D. C. Pieri, E. B. Pitman, and M. F. Sheridan, "Digital elevation model uncertainty and hazard analysis using a geophysical flow model," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science*, vol. 468, no. 2142, pp. 1543–1563, 2012. [Online]. Available: http://rspa.royalsocietypublishing.org/content/468/2142/1543.abstract

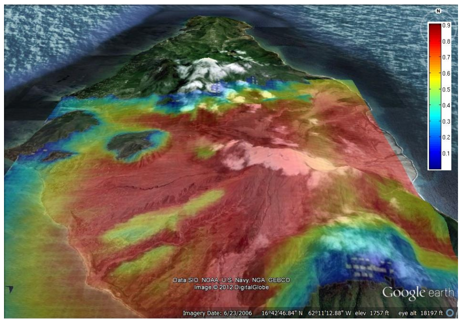

Figure 6. Figure shows the probabilistic hazard map for the island of Montserrat in the Caribbean for flow exceeding 0.2m of depth.

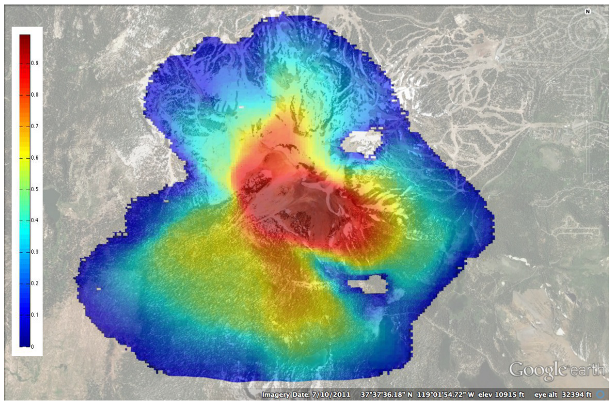

Figure 7. Figure shows the probabilistic hazard map (right) of the region around ”Mammoth” volcano, for flow exceeding 0.2m of depth using 1024 samples generated with the above computing model.