Efficiently Integrating MapReduce-based Computing into a Hurricane Loss Projection Model

Fausto C. Fleites[1], Steve Cocke[2], Shu-Ching Chen[1], Shahid Hamid[3]
[1]*School of Computing and Information Sciences*
[2]*Meteorology Department*
[3]*Department of Finance*
[1,3]*Florida International University, Miami, FL, USA*
[2]*Florida State University, Tallahassee, FL, USA*
[1]{*fflei001,chens*}*@cs.fiu.edu*, [2]*scocke@fsu.edu*, [3]*hamids@fiu.edu*

Abstract

Homeowner insurance is a critical issue for Floridians because of the periodic threat hurricanes pose to Florida. Providing fairness into the rate-making policy process, the state of Florida has developed the Florida Public Hurricane Loss Model (FPHLM), an open, public hurricane risk model to assess the risk of wind damage to insured residential properties. For each input property portfolio, the FPHLM processes a large amount of data to provide expected losses over tens of thousand of years of simulation, for which computational efficiency is of paramount importance. This paper presents our work in integrating the atmospheric component into the FPHLM using MapReduce, which resulted in a highly efficient computing platform for generating stochastic hurricane events on a cluster of computers. The experimental results demonstrate the feasibility of utilizing MapReduce for risk modeling components.

Keywords: hurricane risk modeling, MapReduce, distributed systems.

1. Introduction

Florida is periodically affected by hurricanes which cause significant damage in property and human lives; consequently, homeowner insurance is a critical issue for Floridians. To ensure fair homeowner insurance rates, the state of Florida has developed the Florida Public Hurricane Loss Model (FPHLM) [11][5][6], an open, public hurricane risk model to assess the risk of wind damage to insured residential properties. Since its activation in 2006, the FPHLM has been used more than 500 times by the insurance industry in Florida as well as by insurance regulators.

Developed by a multi-disciplinary team of scientists, as depicted in Fig. 1, the FPHLM's modeling process consists of the following components: an atmospheric component that generates a large number of stochastic storms from a seed of historical events over a large number of years of simulation, an engineering component that links wind information to physical damage creating vulnerability curves for typical building structures, and an actuarial component that estimates expected loss values across the stochastic storm set for an input property portfolio based on the properties' geographic locations, building characteristics, and insurance policy terms. These components are integrated by a computing platform that processes a large amount of data to generate the required results for an input property portfolio. Generally, users of the model need to run portfolios multiple times with varying characteristics, and obtaining the results as as fast as possible allows them to better assess the risk of such portfolios. However, it is a challenge to develop an optimized computing platform for such complex risk models. The difficulty arises from the interaction of the complex components from different disciplines that make up the model and the intense computation required to process each portfolio.

Our goal is this paper is to discuss our work in integrating the atmospheric component into the FPHLM using MapReduce (MR) [7], which resulted in a highly optimized platform capable of efficiently generating stochastic storm sets on a cluster of computers. The literature of not only hurricane risk modeling but also of the more general case of catastrophe (cat) modeling is mostly concentrated on the science behind the components that make up the model. Very few address aspects of the computing platforms supporting such models. Thus, this work provides value in how to integrate catastrophe modeling components into the supporting computing platform using MR.

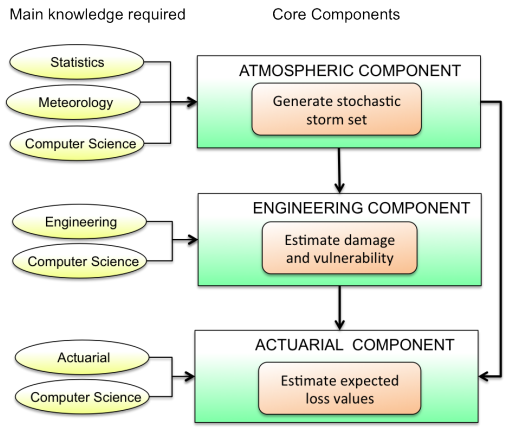

Figure 1. Main components of the FPHLM.

Published articles in the topic are related to (a) improvements in proprietary cat modeling platforms and (b) open-source efforts in risk modeling. Dehne et al. describe their implementation of an earthquake cat modeling system on the Cell Broadband Engine Architecture (Cell/BE) [8]. The system was implemented on a PlayStation 3 whose Cell/BE provides eight co-processing units that accelerate vector operations; in addition, they incorporate other optimization techniques, such as caches, vectorization of the computation, and double-buffering I/O. Mahmood [12] and Dönni [9] report their efforts in porting the computing framework of a re-insurance company to a Grid-based architecture. Mahmood utilized ProActive [4], a Grid middle-ware library, to distribute the computation of the expected losses for the input portfolio and store the results in a database. Dönni focused on improving the data handling of the distributed version of the platform. In the open-source domain, Agora [3], the Alliance for Global Open Risk Analysis, spurs open-source software for multi-hazard risk modeling. Related software are OpenSHA [10] (seismic hazard analysis), OpenSees [2] (earthquake engineering simulation), and OpenRisk [13] (multi-hazard risk modeling). OpenSHA uses Java distributed component technique such as RMI (Remote Method Invocation) to execute remote components; OpenSees provides a flexible and extensible object-oriented architecture that allows the development of sequential and parallel finite element applications for structural analysis; and OpenRisk is object-oriented and web- and GUI-enabled.

The reminder of this paper is organized as follows. Section 2 describes the serial execution of the atmospheric component of the FPHLM. Section 3 details the MR-based design of the component. Section 4 presents the experimental results. Finally, section 5 concludes this work.

2. Atmospheric Component

Over tens of thousand of years of simulation, the atmospheric component generates the storm tracks and wind fields for simulated storms based on stochastic algorithms and random historical initial conditions obtained from the historical record of the Atlantic tropical cyclone basin [11]. For each stochastic storm event, the output of the component consists of 3-second terrain corrected (i.e., taking into consideration the roughness of the terrain) gust wind speeds for each property affected by the event. Fig. 2 depicts the serial execution of an input property portfolio through the FPHLM. Red arrows represent online control flow (i.e., flow executed for each input portfolio), whereas black arrows represent offline control flow (i.e., flow executed once per version of the model).

An input portfolio consists of a set of building properties, which are described by their street addresses, building characteristics (e.g., year of construction, construction type, number of stories, etc.), coverage values, and insurance policy terms. Upon reception, the portfolio is geocoded to obtain the geographic locations (latitude and longitude coordinates) of the properties from their street addresses. Geocoding is the process of locating the geographic coordinates of a set of street addresses.

The atmospheric component is further divided into the Storm Forecast, Wind Field, and Wind Speed Correction components. The Storm Forecast component generates the stochastic storm tracks and intensities based on initial conditions and empirically derived probability distribution functions. Once a storm is close to within a threshold distance to Florida, the Wind Field component is activated to generate the 1-hour marine surface winds of the storm. These winds are interpolated to 10-minute over a 1km fixed grid that entirely covers Florida at every time step, creating a wind swath for each storm. The wind swaths over the fixed grid are stored into several lookup tables for later access. These lookup tables are referred as “tiles”, where each tile represents a rectangular geographic area of Florida. Since they are independent of the input property portfolio, these two components are executed once per version of the FPHLM.

The estimation of the final winds (i.e., winds at the street level) for the input property portfolio is carried out by the Wind Speed Correction component. This component uses roughness information and the marine surface winds to generate the terrain-corrected 3-second gust winds at the street level, which are subsequently used by Actuarial component of the FPHLM to estimate the expected insured losses for each property. The roughness data are encoded at 90m resolution. In addition, all the roughness data are also divided into the same tiles as the storms’ wind swaths. Firstly, using nearest-neighbor approximation, the Wind Speed Correc-

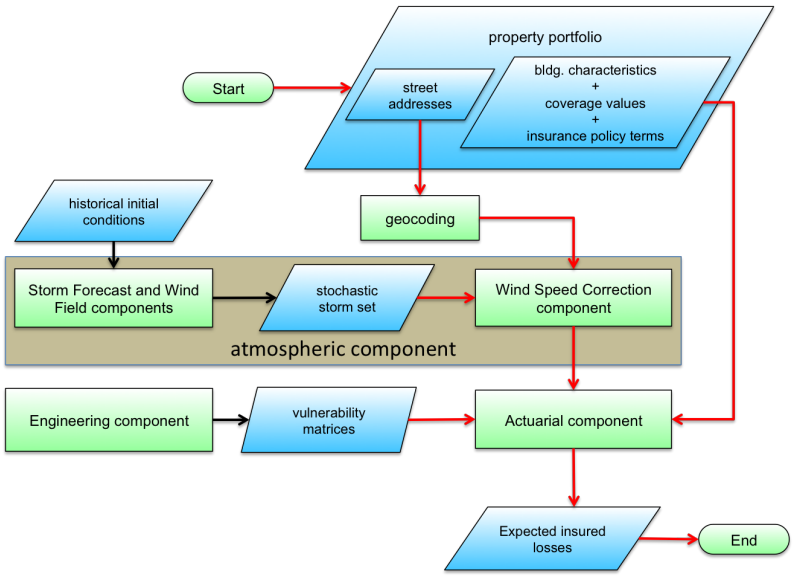

Figure 2. Serial execution of an input property portfolio through the FPHLM.

tion component interpolates the 10-minute marine surface winds over the 1km fixed grid to the resolution of the roughness data. Subsequently, it converts the 10-minute marine winds to 3-second gust winds by calculating the marine drag coefficient, converting the winds from marine to actual or open terrain, calculating the gust factor, and incorporating the effect of sea-land transition in coastal regions. For the purpose of this paper, it suffices to abstract the algorithm of the Wind Speed Correction component as shown in Algorithm 1.

Algorithm 1 $wsc(C, k)$

1: Load R_k // *roughness data corresponding to tile* k
2: Load S_k // *storms' wind swaths of tile* k
3: **for all** $s_i^k \in S_k$ **do**
4: **for all** $c \in C$ **do**
5: **if** c falls within the geographic area of tile k **then**
6: Compute 3-sec gust winds w_i^c for c in s_i^k
7: **end if**
8: **end for**
9: **end for**

The input to the Wind Speed Correction component consists of the geographic coordinate file C and a tile identifier k. The algorithm first loads the roughness data R_k and the storms' wind swaths S_k corresponding to tile k (lines 1 and 2). Subsequently, lines 3-9, it computes the 3-sec gust winds w_i^c for all the coordinates c in C that fall in the geographic area of tile k (line 5) and are affected by the wind swath s_i^k in tile k of the stochastic storm i.

The complete processing of the Wind Speed Correction component consists in executing $wsc(C, k)$ for each tile k. After all tiles have been executed, a "merge" process aggregates all the 3-second gust winds generated for the same storm across all tiles.

3. MapReduce-based Integration

MR is a popular programming model introduced by Google [7] on which distributed applications can be developed to efficiently process large amounts of data. Hadoop [1], its open-source implementation, provides a distributed file system called Hadoop Distributed File System (HDFS) and MR. One appealing feature of MR is that is abstracts programmers from details of parallelization, fault-tolerance, data distribution, and load balancing, which greatly reduces the complexity of developing distributed applications.

A MR program consists of two user-defined functions: a *map* function that is used to process pieces of the input data called input splits, and a *reduce* function used to aggregate the output of invocations of the map function. Both functions use user-defined key-value pairs as input and output. The execution of a MR job consists of two phases: a map phase and a reduce phase, and each phase is divided into tasks. A map task is an invocation of the map function to process an input split, and a reduce task calls the reduce function to aggregate map output records. The MR framework sorts the map-generated key-value pairs by key, copies them to the node where the reduce operation is performed, and merges the sorted pairs from all the map tasks. Hence, each invocation of the reduce function receives as input a

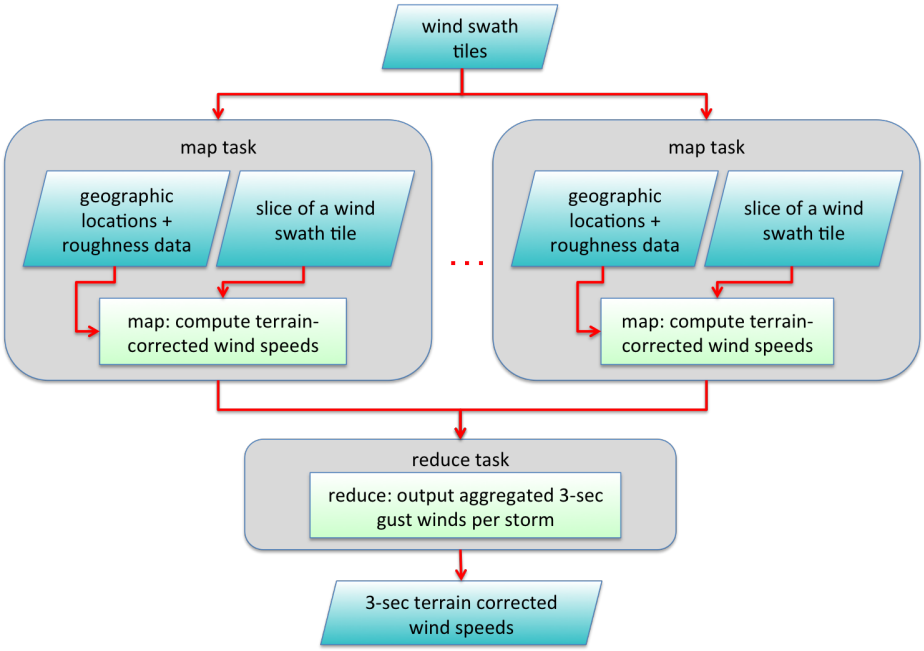

Figure 3. MapReduce-based computing framework for the atmospheric component.

map-generated key and the list of all the values (across all map tasks) that correspond to the key.

Since both the Storm Forecast and Wind Field components are only run once per version of the model, the Wind Speed Correction component is the one that is integrated into the FPHLM via MR. It is realized by one MR job as depicted in Fig. 3. Instead of the geographic coordinates, the input to the MR job consists of the wind swath tiles. The supporting rationale is the geographic locations and storms' wind swaths are the only two datasets that are iterated over, and the latter is the one that is large-scale and can be divided into input splits by the MR framework. Consequently, side data (i.e., data loaded by the MR tasks to process input splits) to the MR job consists of the geographic locations and the roughness data.

Algorithm 2 $setup(I)$

1: Let k be the tile id of I
2: **if** k differ from the previously loaded tile **then**
3: Load R_k // *roughness data corresponding to tile* k
4: Load C_k // *coordinate file corresponding to tile* k
5: **end if**

The overall algorithm of the map function of the MR job is depicted in algorithms 2 and 3.

The $setup$ function is called by the MR framework before processing the records of an input split (i.e., a subset of the storms' wind swaths in a tile). It receives as input an input split I. In line 1, the $setup$ function obtains the id k of I. This can be accomplished by, for example, encoding the id of the tile in the file names of the wind swath tiles. Subsequently, lines 2-5, the $setup$ function loads the roughness data corresponding to tile k (line 3), as well as the list of geographic coordinates C_k that fall in the geographic area of tile k (line 4). Before executing the MR job, a pre-processing script splits the input geographic coordinates file C into k files, where each file C_k correspond to tile k. The loaded R_k and C_k are subsequently used by each invocation of the map function on the input split. This is accomplished by re-utilizing Java virtual machines, a feature provided by the MR framework.

Algorithm 3 map$\left(s_i^k\right)$

1: **for all** $c \in C_k$ **do**
2: Compute 3-sec gust winds w_i^c for c in s_i^k
3: Add w_i^c to U
4: **end for**
5: output $[i, U]$

The map function is called to process a record of an input split, where a record consists of s_i^k, i.e., the storm i's wind swath in a tile k. Having pre-loaded C_k in the $setup$ function, the map function basically iterates over all the geographic coordinates c in C_k (line 1) and computes the 3-second gust winds w_i^c for c in storm i (line 2). Now, since all the 3-second winds speeds have to be aggregated per storm as output of the Wind Speed Correction component, the map function locally aggregates in U the computed winds

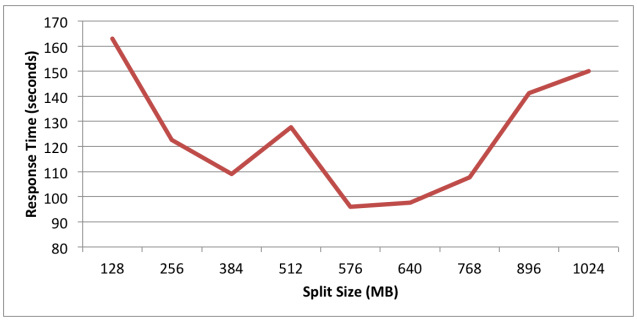

Figure 4. Response time as block size increases.

(line 3) and outputs in line 5 only one key-value pair, in which the key corresponds to the storm's id i and the value is the list of 3-second gust winds U.

The $reduce$ function is very simple. Since the MR framework will aggregate all the map-generated values across all map tasks to the same map-generated key, the $reduce$ function receives as input a storm id i and the list of all the 3-second gust winds that correspond to the storm. Hence, the $reduce$ function just outputs the input key-value pair to HDFS.

4. Experiments

To evaluate the MR-based design of the atmospheric component, we conducted experiments that assess the scalability of the new design. The code was implemented in Java 1.6 and executed in Hadoop 1.0.1. The cluster on which the code was executed consisted of 5 nodes that allow the concurrent execution of 58 MR tasks simultaneously, considering the number of CPU cores in each node and available main memory. Each MR task was assigned 4GB of memory. The maximum number of map tasks that can execute concurrently is referred to as the map task capacity. Moreover, for each property portfolio size used in the following experiments, the same amount of coordinates were randomly generated from each tile, ensuring the same amount of geographic coordinates fell in each.

The purpose of the first experiment is to obtain the optimal split size that, with a fixed task capacity of 58, will yield the fastest response time. When the task capacity is fixed, obtaining a good split size is important to keep all the available map slots occupied all the time. For example, with the given task capacity, if the number of spawned map tasks is 60, then 56 map slots will be unused while the last 2 tasks are processed. On the other hand, if the number of spawned map tasks is 116, then there will be no idle map slots in the cluster, and the response time for the component will be faster.

Table 1. Number of map tasks spawned for minimum split sizes of Fig. 4.

Split Size (MB)	Num. of Map Tasks
128	454
256	235
384	158
512	122
576	109
640	99
768	83
896	79
1024	63

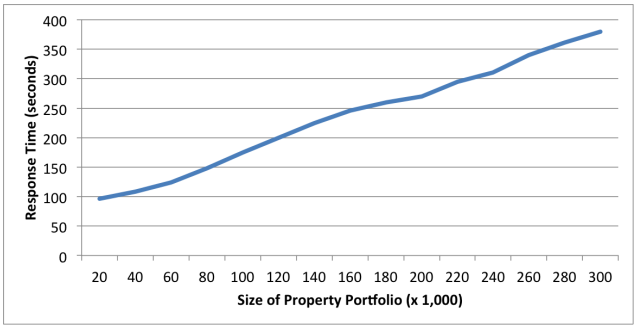

Figure 5. Response time as size of portfolio increases.

The results for this experiment are shown in Fig. 4, which depicts the response time in seconds as the split size increases from 128MB up to 1024MB using in all cases two reduce tasks. Table 1 contains the actual number of map tasks spawned by the split sizes used in this experiment. The size of the property portfolio used in this experiment was 20,000. A split size of 576MB yields the fastest response time with 96 seconds. It provides the best combination of task capacity utilization and number of spawned map tasks. Not utilizing too many map tasks is important for the performance of the component given that it takes a few seconds to setup and schedule a task. The reason for the peak at 512 is the ratio of spawned map tasks to task capacity was 2.1, causing a significant amount of idle map slots.

The second experiment measures the response time in seconds of the MR-based atmospheric component as the size of the property portfolio increases from 20,000 properties through 300,000 properties, in increments of 20,000. All runs in this experiment use two reduce tasks. Instead of the default split size of 64MB, the split size of this experiment was set to 512MB, which provided the fastest response time in the previous experiment. The results are as

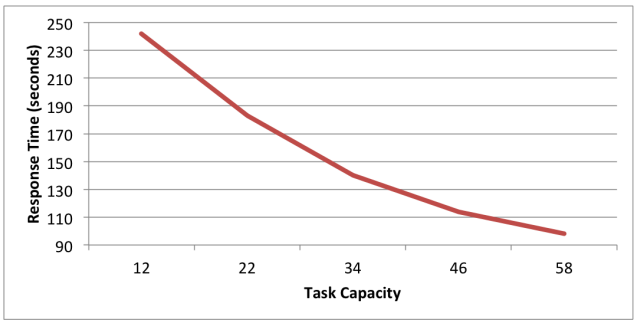

Figure 6. Response time as task capacity increases.

expected, as shown in Fig. 5. The input to the MR job is constant as it consists of the wind swaths tiles, and the geographic coordinates are loaded as side data and looped over in the map function. Hence there should a linear relationship between the number of geographic locations and the response time.

The third experiment measures the response time as the cluster increases its task capacity by adding nodes. Fig. 6 depicts the results. Each node added 12 tasks to the capacity, except for the second one that added 10. The results are as expected. There is an inverse linear relationship between the response time and the task capacity. Two reduce tasks were utilized.

5. Conclusions

This paper has presented our work in using MapReduce in the Florida Public Hurricane Loss Model (FPHLM) to integrate the latter's atmospheric component. The FPHLM is a multi-disciplinary, open, public hurricane risk model that assesses wind damage to insured residential properties in Florida. Users of the model include the insurance industry and insurance regulators, which need the output of the FPHLM as fast as possible to be able to run property portfolios multiple times and better assess their risk. The atmospheric component of the FPHLM generates storm tracks and wind fields for simulated storms over tens of thousand of years of simulation. For an input property portfolio, the output atmospheric component consists of the 3-second terrain-corrected gust wind speeds the properties have on each of the stochastically simulated storms, making the execution of the atmospheric component computationally intensive. Our MapReduce-based integration of the atmospheric component into the FPHLM consists of one MapReduce job that provides an scalable and efficient computing framework. The experimental results validate the scalability of the MapReduce-based component, demonstrating the feasibility of utilizing MapReduce to scale the performance of risk modeling components.

Acknowledgments

This work is partially supported by the Florida Office of Insurance Regulation (OIR) under the "Hurricane Loss Projection Model" project. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the OIR.

References

[1] Hadoop. http://hadoop.apache.org, May.

[2] Opensees. http://opensees.berkeley.edu, 2006.

[3] Agora: Alliance for global open risk analysis. http://www.risk-agora.org, Aug. 2007.

[4] Proactive parallel suite. http://proactive.activeeon.com/index.php, 2013.

[5] S.-C. Chen, M. Chen, N. Zhao, S. Hamid, K. Chatterjee, and M. Armella. Florida public hurricane loss model: Research in multi-disciplinary system integration assisting government policy making. *Special Issue on Building the Next Generation Infrastructure for Digital Government, Government Information Quarterly*, 26(2):285–294, 2009.

[6] S.-C. Chen, S. Gulati, S. Hamid, X. Huang, L. Luo, N. Morisseau-Leroy, M. D. Powell, C. Zhan, and C. Zhang. A web-based distributed system for hurricane occurrence projection. *Softw. Pract. Exper.*, 34(6):549–571, May 2004.

[7] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.

[8] F. Dehne, G. Hickey, A. Rau-Chaplin, and M. Byrne. Parallel catastrophe modelling on a cell processor. In *Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '09, pages 24–31, Riverton, NJ, USA, 2009. IBM Corp.

[9] D. Dönni. Modification of the data handling to grid-enable reinsurance natural catastrophe calculations, 2007. Internship report.

[10] E. Field, T. Jordan, and C. Cornell. Opensha: A developing community-modeling environment for seismic hazard analysis. *Seismological Research Letters*, 77(4):406–419, 2003.

[11] S. Hamid, B. M. G. Kibria, S. Gulati, M. Powell, B. Annane, S. Cocke, J.-P. Pinelli, K. Gurley, and S.-C. Chen. Predicting losses of residential structures in the state of florida by the public hurricane loss evaluation model. *Statistical Methodology*, 7(5):552–573, 2010.

[12] H. Mahmood. Application of distributed computing to a reinsurance natural catastrophe calculation software. Master's thesis, Royal Institute of Technology (KTH), Stockholm, Sweden, 2006.

[13] K. Porter and C. Scawthorn. Openrisk: Open-source risk software and access for the insurance industry. In *First International Conference on Asian Catastrophe Insurance (ICACI)*, 2007.