Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 18 (2013) 2317 – 2326

Procedia Computer Science

International Conference on Computational Science, ICCS 2013

A MapReduce Framework for Analysing Portfolios of Catastrophic Risk with Secondary Uncertainty

A. Rau-Chaplin[a], B. Varghese[a,*], Z. Yao[a]

[a] *Faculty of Computer Science, Dalhousie University, Halifax, Nova Scotia, Canada*

Abstract

The design and implementation of an extensible framework for performing exploratory analysis of complex property portfolios of catastrophe insurance treaties on the Map-Reduce model is presented in this paper. The framework implements Aggregate Risk Analysis, a Monte Carlo simulation technique, which is at the heart of the analytical pipeline of the modern quantitative insurance/reinsurance pipeline. A key feature of the framework is the support for layering advanced types of analysis, such as portfolio or program level aggregate risk analysis with secondary uncertainty (i.e. computing Probable Maximum Loss (PML) based on a distribution rather than mean values). Such in-depth analysis is not supported by production-based risk management systems since they are constrained by hard response time requirements placed on them. On the other hand, this paper reports preliminary experimental results to demonstrate that in-depth aggregate risk analysis can be realized using a framework based on the MapReduce model.

Keywords: MapReduce model, secondary uncertainty, risk modelling, aggregate risk analysis

1. Introduction

At the heart of the analytical pipeline of the modern quantitative insurance/reinsurance company are *productions systems* that perform Aggregate Risk Analysis on portfolios of complex property catastrophe insurance treaties (for example, the Risk Management Solutions reinsurance platform [1], and the research reported in [2, 3, 4, 5]). Such systems typically perform a small set of core analytical functions and are highly optimized for speed, reliability, and regulatory compliance. Production systems often achieve very high performance, but at a cost in that (i) they ruthlessly aggregate results up to the entire portfolio level making detailed analysis of sub-components of the portfolio difficult or almost impossible and (ii) they exploit specialized software-hardware design methodologies that make them difficult or impossible to extend.

In this paper, the design and implementation of an extensible framework for performing ad hoc analysis of portfolios of catastrophic risk based on the MapReduce programming model [6, 7, 8] using the Hadoop platform [9, 10, 11] is explored. The goal is to employ the framework to facilitate an environment for analysts in which they can (i) explore risk management questions not anticipated by the designers of production systems, (ii) perform a more in-depth analysis at a finer level of detail than what is supported by the production system, and (iii) prototype significant extensions which provides an insight into the portfolio on a monthly or quarterly basis (this may be too computationally expensive for production use).

*Corresponding author
E-mail address: varghese@cs.dal.ca.

doi:10.1016/j.procs.2013.05.403

Aggregate risk analysis can be used to compute Probable Maximum Loss (PML) [12, 13] and the Tail Value-at-Risk (TVAR) [14, 15] metrics for an entire portfolio. However, in addition the analysts may want to compute (a) Portfolio or Program level Probable Maximum Loss (PML) analysis taking into account secondary uncertainty, that is computing PMLs based on a distribution rather than just a mean value, (b) Year Loss Table/Return Period Losses by Treaty Line of Business, that is taking a defined portfolio and filtering the Layers by Line of Business (LOB), (c) Year Loss Table/Return Period Losses by Class of Business (CoB), that is taking a defined portfolio and filtering the Layers by CoB, (d) Region Peril filtering, that is taking a loss sets broken down by peril region and analysing just the selected peril regions for specific programs or a set of programs, (e) Iterative Marginals, that is adding/subtracting a specified program to/from a portfolio and computing every combination of marginal for each program, (f) STEP Analysis, that is taking events in the catalogue and using them to make a combine loss distribution for a single event, and (g) Monthly/Weekly Loss Distributions, that is using the portfolio analysis to see the yearly distribution of losses (i.e the portfolio's loss seasonality).

While such in-depth analysis is typically not supported by production systems that have hard response time requirements, this paper explores how it can be realized by a MapReduce framework.

In the remainder of this paper, the design and implementation of the fundamental aggregate risk analysis simulations using MapReduce, and an example of how the calculation of secondary uncertainty can be layered on top of the simulations is performed. Section 2 presents the aggregate analysis, firstly the sequential algorithm followed by the Map-Reduce algorithm. Section 3 shows how to compute secondary uncertainty within the aggregate analysis problem. Section 4 considers the implementation of aggregate analysis on the Apache Hadoop platform. The preliminary results obtained from experiments are reported in Section 5. The paper concludes by presenting areas of future work in Section 6.

2. Aggregate Risk Analysis (ARA)

In this section, firstly the sequential aggregate risk analysis algorithm is presented, followed by the parallel aggregate risk analysis algorithm on the Hadoop Map-Reduce platform. The inputs and the output of ARA are the same. There are three inputs to the ARA algorithm, namely the YET, PF, and a pool of $ELTs$. The YET is the Year Event Table which is the representation of a pre-simulated occurrence of Events E in the form of trials T. Each Trial captures the sequence of the occurrences of Events for a year using time-stamps in the form of event time-stamp pairs. The PF is a portfolio that represents a group of Programs, P, which in turn represents a set of Layers, L that covers a set of $ELTs$ using financial terms. The ELT is the Event Loss Table which represents the losses that correspond to an event based on an exposure (one event can appear over different ELTs with different losses). An eXtended ELT (XELT) contains additional information based on the Event, the independent and correlated standard deviations, the mean and the maximum expected losses for an event to compute secondary uncertainty considered in Section 3.

Two intermediary output of ARA are the Layer Loss Table LLT and the Program Loss Table PLT both consisting Trial-Loss pairs. The final output of ARA algorithm is YLT, which is the Year Loss Table that contains the losses covered by a portfolio.

2.1. Sequential ARA

Algorithm 1 shows the sequential analysis of aggregate risk. The algorithm scan through the hierarchy of the portfolio, PF; firstly through the Programs, P, followed by the Layers, L, then the Event Loss Tables, $ELTs$. Line no. 5-9 shows how the loss associated with an Event in the ELT is computed. For this, the loss, l_E associated with an Event, E is retrieved, after which secondary uncertainty is applied. The computation of secondary uncertainty will be considered in the next section. Contractual financial terms to the benefit of the Layer are applied to the losses and are summed up as l'_E.

In line no. 10 and 11, two Occurrence Financial Terms, namely the Occurrence Retention and the Occurrence Limit are applied to the loss, l'_E and summed up as l_T. The l_T losses correspond to the total loss in one trial. Occurrence Retention refers to the retention or deductible of the insured for an individual occurrence loss, where as Occurrence Limit refers to the limit or coverage the insurer will pay for occurrence losses in excess of the retention. The Occurrence Financial Terms capture specific contractual properties of 'eXcess of Loss' treaties as they apply to individual event occurrences only.

```
   Input  : $YET$, $ELT$ pool, $PF$
   Output: $YLT$
 1 for each Program, P do
 2     for each Layer, L, in P do
 3         for each Trial, T, in YET do
 4             for each Event, E, in T do
 5                 for each ELT covered by L do
 6                     Lookup $E$ in the $ELT$ and find corresponding loss, $l_E$;
 7                     Apply Secondary Uncertainty and Financial Terms to $l_E$;
 8                     $l'_E \leftarrow l'_E + l_E$;
 9                 end
10                 Apply Occurrence Financial Terms to $l'_E$;
11                 $l_T \leftarrow l_T + l'_E$;
12             end
13             Apply Aggregate Financial Terms to $l_T$;
14             Populate Trial-Loss pairs in $LLT$ using $l_T$;
15         end
16     end
17     Sum losses of Trial-Loss pairs in all $LLT$;
18     Populate Trial-Loss pairs in $PLT$;
19 end
20 Aggregate losses of Trial-Loss pairs in $PLT$;
21 Populate $YLT$;
```

Algorithm 1: Pseudo-code for Sequential Aggregate Risk Analysis

In line no. 13 and 14, two Aggregate Financial Terms, namely the Aggregate Retention and the Aggregate Limit are applied to the loss, l_T to produce aggregated loss for a Trial. Aggregate Retention refers to the retention or deductible of the insured for an annual cumulative loss, where as Aggregate Limit refers to the limit or coverage the insurer will pay for annual cumulative losses in excess of the aggregate retention. The Aggregate Financial terms captures contractual properties as they apply to multiple event occurrences. The trial-loss pairs are then used to populate Layer Loss Tables $LLTs$; each Layer is represented using a Layer Loss Table consisting of Trial-Loss pairs.

In line no. 17 and 18, the trial losses are aggregated from the Layer level to the Program level. The losses are represented again as a trial-loss pair and are used to populate Program Loss Tables $PLTs$; each Program is represented using a Program Loss Table.

In line 20 and 21, the trial losses are aggregated from the Program level to the Portfolio level. The trial-loss pairs are populated in the Year Loss Table YLT which represents the output of the analysis of aggregate risk. Financial functions or filters are then applied on the aggregate loss values.

2.2. Map-Reduce ARA

MapReduce is a programming model developed by Google for processing large amount of data on large clusters. A map and a reduce function are adopted in this model to execute a problem that can be decomposed into sub-problems with no dependencies; therefore the model is most attractive for embarrassingly parallel problems. This model is scalable across large number of computing resources. In addition to the computations, the fault tolerance of the execution, for example, handling machine failures are taken care by MapReduce. An open-source software framework that supports the MapReduce model, Apache Hadoop [9, 10, 11], is used in the research reported in this paper.

The MapReduce model lends itself well towards solving embarrassingly parallel problems, and therefore, the analysis of aggregate risk is explored on MapReduce. In the analysis of aggregate risks, the Programs contained in the Portfolio are independent of each other, the Layers contained in a Program are independent of each other and further the Trials in the Year Event Table are independent of each other. This indicates that the problem of analysing aggregate risks requires a large number of computations which can be performed on independent parallel problems.

Another reason of choice for the MapReduce model is that it can handle large data processing for ARA. All Events in the Year Event Table need to be processed for every Layer which accounts for the largeness of the data. For example, consider a Year Event Table comprising one million simulations, which is approximately 30 GB. So for a Portfolio comprising 2 Programs, each with 10 Layers, then the approximate volume of data that needs to be processed is 600GB.

Further MapReduce implementations such as Hadoop provide dynamic job scheduling based on the availability of cluster resources and distributed file system fault tolerance.

Algorithm 2 shows the map-reduce analysis of aggregate risk. The aim of this algorithm is similar to the sequential algorithm in which the algorithm scans through the Portfolio, PF; firstly through the Programs, P, and then through the Layers, L. The first round of MapReduce jobs, denoted as $MapReduce_1$ are launched for all the Layers. The Map function (refer Algorithm 3) scans through all the Event Loss Tables $ELTs$ covered by the Layers L to compute the losses l'_E in parallel for every Event in the ELT. The computations of loss l_T at the Layer level are performed in parallel by the Reduce function (refer Algorithm 4). The output of $MapReduce_1$ is a Layer Loss Table LLT.

```
Input  : $YET$, $ELT$ pool, $PF$
Output: $YLT$

1 forall Programs of P do
2     forall Layers L in P do
3         $LLT \leftarrow MapReduce_1(L, YET)$;
4     end
5 end
6 $YLT \leftarrow MapReduce_2(LLTs)$;
```

Algorithm 2: Pseudo-code for Parallel Aggregate Risk Analysis

The second round of MapReduce jobs, denoted as $MapReduce_2$ are launched for aggregating all the $LLTs$ in each Program to a YLT. Unlike the sequential algorithm no $PLTs$ are generated as the intermediate output as the Reducer can aggregate all the trial-loss pairs from the Layer level to the Portfolio level.

The master node of the cluster of nodes solving a problem partitions the input data to intermediate files effectively splitting the problem into sub-problems. The sub-problems are distributed to the worker nodes by the master node, often referred to as the 'Map' step performed by the Mapper. The map function executed by the Mapper receives as input a $< key, value >$ pair to generate a set of $< intermediate\ key, intermediate\ value >$ pairs. The results of the decomposed sub-problems are then combined by the Reducer referred to as the 'Reduce' step. The Reduce function executed by each Reducer merges the $< intermediate\ key, intermediate\ value >$ pairs to generate a final output. The Reduce function receives all the values corresponding to the same intermediate key.

Algorithm 3 and Algorithm 4 show how parallelism is achieved by using the Map and Reduce functions in a first round at the Layer level in ARA. Algorithm 3 shows the Map function whose inputs are a set of T, E from the YET, and the output is a Trial-Loss pair $< T, l'_E >$ which corresponds to an Event. To estimate the loss, it is necessary to scan through every Event Loss Table ELT covered by a Layer L (line no. 1-5). Similar to the sequential algorithm the loss, l_E associated with an Event, E in ELT is fetched from memory in line no. 2. Secondary uncertainty and contractual financial terms to the benefit of the layer are applied to the losses (line no. 3) to aggregate the losses as l'_E (line no. 4). The loss for every Event in a Trial is emitted as $< T, l'_E >$.

Algorithm 4 shows the Reduce function used in the ARA. The inputs are the Trial T and the set of losses (l'_E) corresponding to that Trial, represented as L'_E, and the output is a Trial-Loss pair $< T, l_T >$. Similar to the sequential algorithm for every loss value l'_E in the set of losses L'_E, the Occurence Financial Terms, namely Occurrence Retention and the Occurrence Limit, are applied to l'_E (line no. 2) and summed up as l_T (line no. 3). The Aggregate Financial Terms, namely Aggregate Retention and Aggregate Limit are applied to l_T (line no. 5). The aggregated loss for a Trial, l_T is emitted as $< T, l_T >$ to populate the Layer Loss Table.

```
Input  : $< T, E >$
Output: $< T, l'_E >$
1 for each $ELT$ covered by $L$ do
2     Lookup $E$ in the $ELT$ and find corresponding loss, $l_E$;
3     Apply Secondary Uncertainty and Financial Terms to $l_E$;
4     $l'_E \leftarrow l'_E + l_E$;
5 end
6 Emit($T$, $l'_E$)
```

Algorithm 3: Pseudo-code for Map function in $MapReduce_1$ of Aggregate Risk Analysis

```
Input  : $T, L'_E$
Output: $< T, l_T >$
1 for each $l'_E$ in $L'_E$ do
2     Apply Occurrence Financial Terms to $l'_E$;
3     $l_T \leftarrow l_T + l'_E$;
4 end
5 Apply Aggregate Financial Terms to $l_T$;
6 Emit($T$, $l_T$)
```

Algorithm 4: Pseudo-code for Reduce Function in $MapReduce_1$ of Aggregate Risk Analysis

Algorithm 5 and Algorithm 6 show how parallelism is achieved by using the Map and Reduce functions in a second round for aggregating all Layer Loss Tables to produce the YLT in ARA (the operations in the sequential algorithm are shown in line no. 17, 18, 20 and 21). Algorithm 5 shows the Map function whose inputs are a set of Layer Loss Tables $LLTs$, and the output is a Trial-Loss pair $< T, l_T >$ which corresponds to the Layer-wise loss for Trial T.

Algorithm 6 shows the Reduce function whose inputs are a set of losses corresponding to a Trial in all Layers L_T, and the output is a Trial-Loss pair $< T, l'_T >$ which is an entry to populate the final output of ARA, the Year Loss Table YLT. The function sums up all trial losses l_T across all Layers to produce a portfolio-wise aggregate loss l'_T.

```
Input  : $LLTs$
Output: $< T, l_T >$
1 for each $T$ in $LLT$ do
2     Emit($< T, l_T >$)
3 end
```

Algorithm 5: Pseudo-code for Map function in $MapReduce_2$ of Aggregate Risk Analysis

```
Input  : $< T, L_T >$
Output: $< T, l'_T >$
1 for each $l_T$ in $L_T$ do
2     $l'_T \leftarrow l'_T + l_T$
3 end
4 Emit($< T, l'_T >$)
```

Algorithm 6: Pseudo-code for Reduce function in $MapReduce_2$ of Aggregate Risk Analysis

3. Applying Secondary Uncertainty

In this section, the methodology to compute secondary uncertainty is presented; this method heavily draws on industry wide practices. The inputs and their representations are firstly presented, followed by the sequence of steps for combining independent and correlated standard deviations, and finally computing the losses which are calculated based on the Beta distribution.

3.1. Inputs

There are six inputs required for computing secondary uncertainty, which are:

i. Program-and-Event-Occurrence-Specific random number, denoted as $z_{(Prog,E)} = P_{(Prog,E)} \in U(0,1)$. Each Event occurrences across different Programs have different random numbers, obtained from YET.
ii. Event-Occurrence-Specific random number, denoted as $z_{(E)} = P_{(E)} \in U(0,1)$. Each Event occurrence across different Programs have the same random number obtained from XELT.
iii. Mean loss, denoted as μ_L obtained from XELT.
iv. Independent standard deviation of loss, denoted as σ_I, which represents the variance within the event-loss distribution obtained from XELT.
v. Correlated standard deviation of loss, denoted as σ_C, which represents the error of the event-occurrence dependencies obtained from XELT.
vi. Maximum expected loss, denoted as $Loss_{max}$ obtained from XELT.

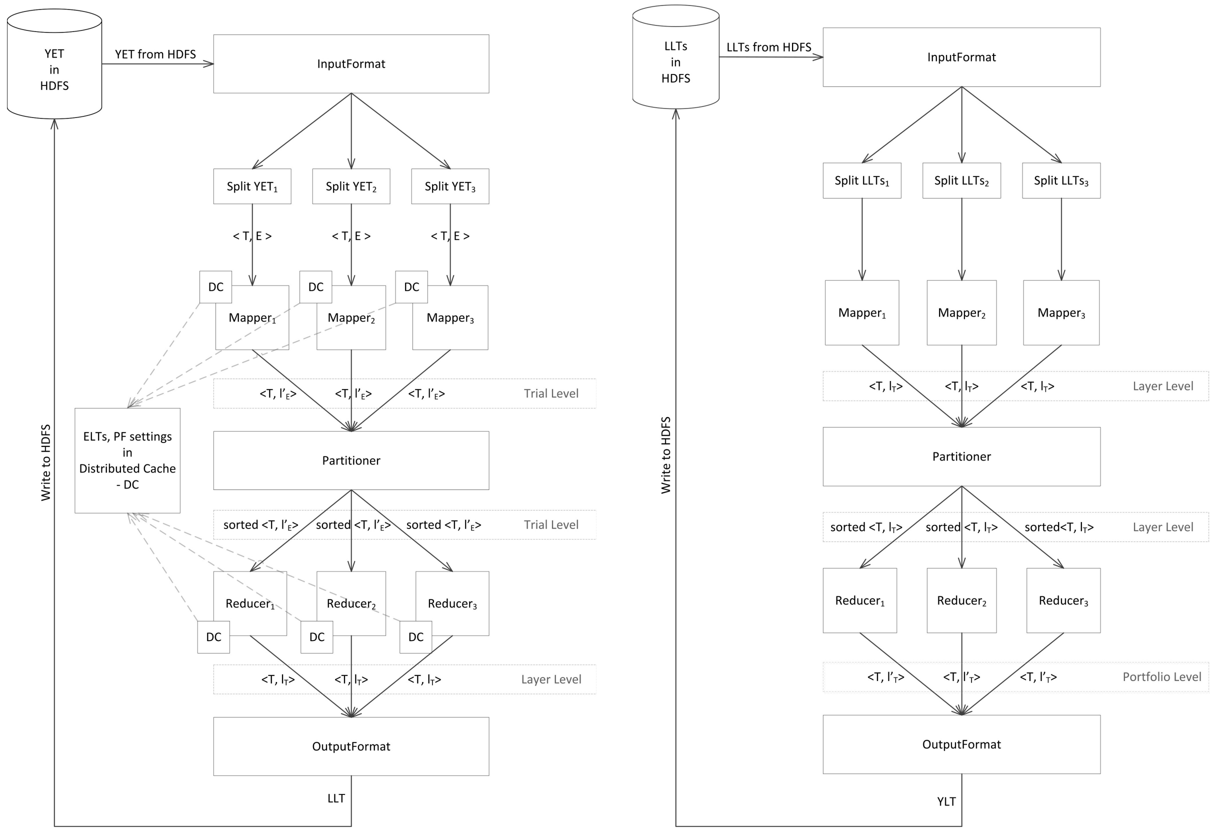
3.2. Steps for combining standard deviation

Given the above inputs, the independent and correlated standard deviations need to be combined to reduce the error in estimating the loss value associated with an event. For this, firstly, the raw standard deviations is produced as $\sigma = \sigma_I + \sigma_C$. Secondly, the probabilities of occurrences, $z_{(Prog,E)}$ and $z_{(E)}$ are transformed from uniform distribution to normal distribution using, $f(x;\mu,\sigma^2) = \int_{-\infty}^{x} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \mathrm{d}x$. This is applied to the probabilities of event occurrences as $v_{(Prog,E)} = f(z_{(Prog,E)};0,1) \in N(0,1)$ and $v_{(E)} = f(z_{(E)};0,1) \in N(0,1)$. Thirdly, the linear combination of the transformed probabilities of event occurrences and the standard deviations is computed as $LC = v_{(Prog,E)}\left(\frac{\sigma_I}{\sigma}\right) + v_{(E)}\left(\frac{\sigma_C}{\sigma}\right)$. Then the normal random variable is computed, fourthly, as $v = \frac{LC}{\sqrt{\left(\frac{\sigma_I}{\sigma}\right)^2 + \left(\frac{\sigma_C}{\sigma}\right)^2}}$. Finally, the normal random variable is transformed from normal distribution to uniform distribution as $z = \Phi(v) = F_{Norm}(v) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{v} e^{\frac{-t^2}{2}} \mathrm{d}t$.

The model used above for combining the independent and correlated standard deviations represents two extreme cases. The first case in which $\sigma_I = 0$ and the second case in which $\sigma_C = 0$. The model also ensures that the final random number, z, is drawn based on both the independent and correlated standard deviations.

3.3. Loss Calculation based on Beta distribution

The loss is calculated based on the Beta distribution as fitting such a distribution allows the representation of risks quite accurately. The Beta distribution is a two parameter distribution, with an upper bound for the standard deviation, and after normalising in the model above, three parameters are used. In the Beta-distribution the standard deviation, mean, alpha and beta are defined as $\sigma_\beta = \frac{\sigma}{Loss_{max}}$, $\mu_\beta = \frac{\mu_L}{Loss_{max}}$, $\alpha = \mu_\beta\left(\left(\frac{\sigma_{\beta max}}{\sigma_\beta}\right)^2 - 1\right)$, and $\beta = (1-\mu_\beta)\left(\left(\frac{\sigma_{\beta max}}{\sigma_\beta}\right)^2 - 1\right)$. An upper bound is set to limit the standard deviation using $\sigma_{\beta_{max}} = \sqrt{\mu_\beta(1-\mu_\beta)}$; if $\sigma_\beta > \sigma_{\beta_{max}}$, then $\sigma_\beta = \sigma_{\beta_{max}}$. For numerical purpose in the algorithm a value very close to $\sigma_{\beta_{max}}$ is chosen. The estimated loss is then obtained by $Loss = Loss_{max} * PDF_{beta}(z;\alpha,\beta)$, $PDF_{beta}(z;\alpha,\beta) = \int_{-\infty}^{z} \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} z^{\alpha-1}(1-z)^{\beta-1}$, and $\Gamma(z)$ is the gamma function. Therefore, $Loss = Loss_{max} * \frac{1}{B(\alpha,\beta)} z^{\alpha-1}(1-z)^{\beta-1}$, where B is the normalisation constant.



(a) First MapReduce round (b) Second MapReduce round

Fig. 1: MapReduce rounds in the Hadoop implementation of Aggregate Risk Analysis

4. Apache Hadoop Implementation

In this section, the experimental platform and the implementation of MapReduce ARA are presented. The experimental platform is a heterogeneous cluster comprising (a) a master node which is an IBM blade of two XEON 2.67GHz processors comprising six cores, memory of 20 GB per processor and a hard drive of 500GB with an additional 7TB RAID array, and (b) six worker nodes each with an Opteron Dual Core 2216 2.4GHz processor comprising four cores, memory of 4GB RAM and a hard drive of 150GB (b). The nodes are connected via Infiniband.

Apache Hadoop, an open-source software framework is used for implementing the MapReduce ARA [9, 10, 11]. Other available frameworks [16, 17] require the use of additional interfaces, commercial or web-based, for deploying an application and were therefore not chosen.

The Hadoop framework works in the following way for a MapReduce round. First of all, the data files from the Hadoop Distributed File System (HDFS) is loaded using the `InputFormat` interface. HDFS provides a functionality called distributed cache for distributing small data files which are shared by the nodes of the cluster. The distributed cache provides local access to the shared data. The `InputFormat` interface specifies the input the Mapper and splits the input data as required by the Mapper. The `Mapper` interface receives the partitioned data and emits intermediate key-value pairs. The `Partitioner` interface receives the intermediate key-value pairs and controls the partitioning of these keys for the `Reducer` interface. Then the `Reducer` interface receives the partitioned intermediate key-value pairs and generates the final output of this MapReduce round. The output is received by the `OutputFormat` interface and provides it back to HDFS.

The input data for MapReduce ARA which are the Year Event Table YET, the pool of Event Loss Table ELT

and the Portfolio PF specification are stored on HDFS. The master node executes Algorithm 2 to generate the Year Loss Table YLT which is again stored on the HDFS. The two MapReduce rounds are illustrated in Figure 1.

In the first MapReduce round the `InputFormat` interface splits the YET based on the number of Mappers specified for the MapReduce round. The Mappers are configured such that they also receive the $ELTs$ covered by one Layer which are contained in the distributed cache. The Mapper applies secondary uncertainty and Financial Terms to the losses. In this implementation combining the $ELTs$ is considered for achieving fast lookup. A typical ELT would contain entries for an Event ID and related loss information. When the $ELTs$ are combined they contain an Event ID and the loss information related to all the individual $ELTs$. This reduces the number of lookups for retrieving loss information related to an Event when the Events in a Trial contained in the YET are scanned through by the Mapper. The Mapper emits a trial-Event Loss pair which is collected by the Partitioner. The Partitioner delivers the trial-Event Loss pairs to the Reducers; one Reducer gets all the trial-Event Loss pairs related to a specific trial. The Reducer applies the Occurrence Financial and Aggregate Financial Terms to the losses emitted to it by the Mapper. Then the `OutputFormat` writes the output of the first MapReduce round as Layer Loss Tables LLT to the HDFS.

In the second MapReduce round the `InputFormat` receives all the $LLTs$ from HDFS. The `InputFormat` interface splits the set of $LLTs$ and distributes them to the Mappers. The `Mapper` interface emits Layer-wise Trial-Loss pairs. The `Partitioner` receives all the Trial-Loss pairs and partitions them based on the Trial for each Reducer. The `Reducer` interface uses the partitioned Trial-Loss pairs and combines them to Portfolio-wise Trial-Loss pairs. Then the `OutputFormat` writes the output of the second MapReduce round as a Year Loss Table YLT to the HDFS.

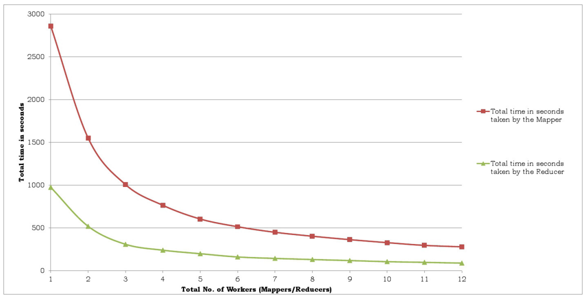
5. Preliminary Results

MapReduce ARA experiments were performed for one Portfolio comprising one Program and one Layer and sixteen Event Loss Tables. The Year Event Table has 100,000 Trials, with each Trial comprising 1000 Events. The experiments are performed for up to 12 workers as there are 12 cores available on the cluster employed for the experiments. The results for the two MapReduce rounds are considered in this section.

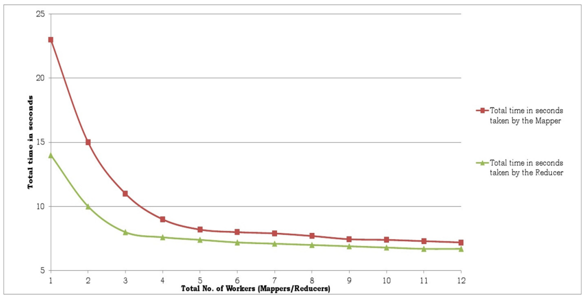
The graph shown in Figure 2 represents the total time taken in seconds by the workers (Mappers and Reducers) of the first MapReduce rounds ($MapReduce_1$) of Algorithm 2. There is close to 100% efficiency when 2 workers are employed, but the performance deteriorates beyond the use of two workers on the cluster employed. The best time obtained for MapReduce is on 12 workers taking a total of 370 seconds, with 280 seconds for the Mapper and 90 seconds for the Reducer. For both the Mappers and the Reducers it is observed that over half the total time is taken for local I/O operations. In the case of the Mapper the mathematical computations take only 1/4th the total time, and the total time taken for data delivery from the HDFS to the `InputFormat`, and from the `InputFormat` to the `Mapper` and from the `Mapper` to the `Partitioner` is only $1/4^{th}$ the total time. In the case of the Reducer the mathematical computations take $1/3^{rd}$ the total time, whereas the total time taken for data delivery from the Partitioner to the Reducer, from the Reducer to the OutputFormat, and from the OutputFormat to HDFS is nearly $1/6^{th}$ the total time. This indicates that the local I/O operations on the cluster employed is expensive though the performance of Hadoop is exploited for both computations and for large data delivery. The two graphs shown in Figure 3 presents the relative speedup of the Mapper and Reducer in the first MapReduce round.

The graph shown in Figure 3 represents the total time taken in seconds by the workers (Mappers and Reducers) of the second MapReduce rounds ($MapReduce_2$) of Algorithm 2. The performance is poor on the cluster employed, and the best time obtained for MapReduce is on 12 workers taking a total of 13.9 seconds, with 7.2 seconds for the Mapper and 6.7 seconds for the Reducer. In this case the I/O overheads and the worker initialisation overheads are large. The two graphs shown in Figure 4 presents the relative speedup of the Mapper and Reducer in the second MapReduce round.

In summary, the results indicate that while there is scope for achieving speedup on mathematical computations and data delivery within the Hadoop system, there seems to be a large overhead for the local I/O operations on the workers. This large overhead is due to the bottleneck of the connectivity between the workers, and the latency in reading data from local drives. However, the trade-off can be minimised if larger input data is employed. The results indicate that the Hadoop implementation of Aggregate Risk Analysis has scope for efficient data delivery

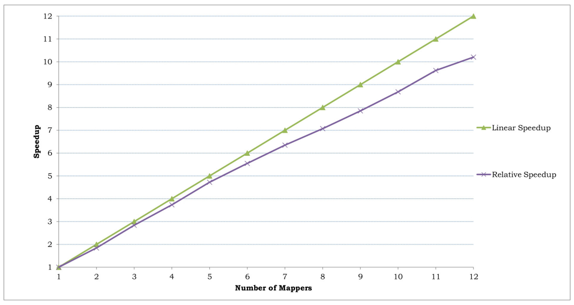


(a) First round

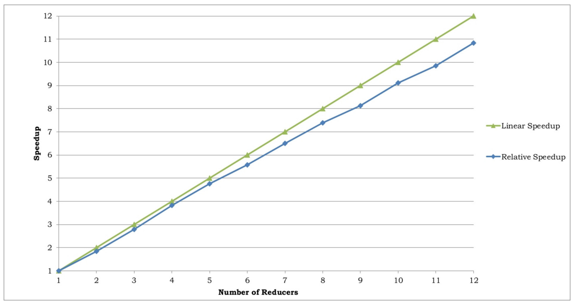


(b) Second round

Fig. 2: Total number of workers vs time taken by the Mapper and Reducer of the MapReduce rounds in Algorithm 2

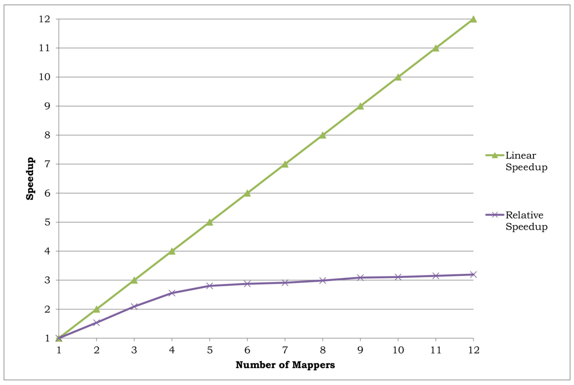


(a) Mapper

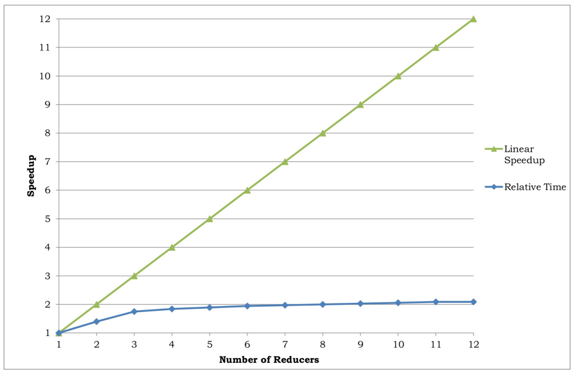


(b) Reducer

Fig. 3: Speedup achieved on the first round of MapReduce



(a) Mapper



(b) Reducer

Fig. 4: Speedup achieved on the second round of MapReduce

and effective mathematical computations. Efforts need to be made towards reducing the I/O overhead to exploit the full benefit of the Hadoop MapReduce model.

6. Conclusion

This paper has proposed a design of an extensible framework to facilitate ad hoc analysis of catastrophic risk-based portfolios. Such an extensible framework can be used for performing analysis of portfolios by taking into account the finer level of detail which is not supported by production-based risk management systems. The proposed framework considers the aggregate risk analysis algorithm and supports the layering of in-depth analysis on top of the basic algorithm that can capture finer level of detail of the Portfolio, Program and Layer levels. In this paper, the consideration of secondary uncertainty while computing the Probable Maximum Loss (PML) adds a layer on the basic aggregate risk analysis algorithm. The finer level of detail is captured by not just considering mean values of losses but a distribution of losses. The proposed framework has been implemented using the MapReduce model on the Apache Hadoop platform. The implementation demonstrates how the calculation of secondary uncertainty can be layered on top of the simulations performed by the basic aggregate risk analysis algorithm. Preliminary results obtained from experiments show that in-depth aggregate risk analysis can be realized using a framework based on the MapReduce model.

In the future, other examples of layering finer level of detail on the aggregate risk analysis algorithm will be considered. Immediate efforts will be made to optimise the implementation for reducing the local I/O overheads to achieve further speedup.

References

[1] RMS Reinsurance Platform, `http://rms.com/`.
[2] R. R. Anderson, W. Dong, Pricing catastrophe reinsurance with reinstatement provisions using a catastrophe model (1998) 303–322.
[3] A. K. Bahl, O. Baltzer, A. Rau-Chaplin, B. Varghese, Parallel simulations for analysing portfolios of catastrophic event risk, in: Workshop of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2012.
[4] W. Dong, H. Shah, F. Wong, A rational approach to pricing of catastrophe insurance 12 201–218.
[5] G. G. Meyers, F. L. Klinker, D. A. Lalonde, The aggregation and correlation of reinsurance exposure (2003) 69–152.
[6] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, Communications of the ACM 51 (1) (2008) 107–113.
[7] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, R. Sears, Mapreduce online, Tech. Rep. UCB/EECS-2009-136, EECS Department, University of California, Berkeley (Oct 2009).
URL `http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-136.html`
[8] K.-H. Lee, Y.-J. Lee, H. Choi, Y. D. Chung, B. Moon, Parallel data processing with mapreduce: A survey, SIGMOD Record 40 (4) (2011) 11–20.
[9] T. White, Hadoop: The Definitive Guide, 1st Edition, O'Reilly Media, Inc., 2009.
[10] Apache Hadoop Project, `http://hadoop.apache.org/`.
[11] K. Shvachko, K. Hairong, S. Radia, R. Chansler, The hadoop distributed file system, in: 26th IEEE Symposium on Mass Storage Systems and Technologies, 2010, pp. 1 –10.
[12] G. Woo, Natural catastrophe probable maximum loss, British Actuarial Journal 8.
[13] M. E. Wilkinson, Estimating probable maximum loss with order statistics, in: Casualty Actuarial Society Forum, 1982, pp. 195–209.
[14] A. A. Gaivoronski, G. Pflug, Value-at-risk in portfolio optimization: Properties and computational approach, Journal of Risk 7 (2) 1–31.
[15] P. Glasserman, P. Heidelberger, P. Shahabuddin, Portfolio value-at-risk with heavy-tailed risk factors, Mathematical Finance 12 (3) 239–269.
[16] Amazon Elastic MapReduce (Amazon EMR, `http://aws.amazon.com/elasticmapreduce/`.
[17] Google MapReduce, `https://developers.google.com/appengine/docs/python/dataprocessing/overview`.