High Performance Risk Aggregation: Addressing the Data Processing Challenge the Hadoop MapReduce Way

A. Rau-Chaplin, B. Varghese[1], Z. Yao
Faculty of Computer Science, Dalhousie University
Halifax, Nova Scotia, Canada
E-mail: {arc, varghese, yao}@cs.dal.ca

***Abstract*—Monte Carlo simulations employed for the analysis of portfolios of catastrophic risk can benefit from platforms that process large volumes of data by exploiting parallelism. To achieve this an algorithm for the analysis of aggregate risk is proposed and implemented using the MapReduce model on the Apache Hadoop framework. An evaluation of the performance of the algorithm indicates that the Hadoop MapReduce model offers a feasible platform for processing large data. An aggregate simulation of 100,000 trials with 1000 catastrophic events per trial on a typical exposure set and contract structure is performed on multiple worker nodes in about 6 minutes. The result indicates the scope and feasibility of MapReduce for tackling the data problem in the analysis of aggregate risk.**

***Keywords*-hadoop mapreduce; risk aggregation; risk analysis; data processing; high-performance analytics**

I. Introduction

In the domain of large-scale computational analysis of risk, large amounts of data are rapidly processed and millions of simulations are quickly performed (for example, [1], [2], [3]). This can be achieved by efficient data management and exploitation of parallelism within simulations. Therefore, the domain inherently opens avenues to exploit the synergy that can be achieved by bringing together state-of-the-art techniques in data management and high-performance computing. There is limited research on aggregate analysis of risks [5], [6], [7] using high-performance computing. Therefore, the research reported in this paper is motivated towards exploring a means to facilitate high-performance data processing and management for aggregate analysis of risk; in this context the MapReduce model [4], [8], [9] is used for achieving high-performance aggregate analysis of risks.

The aggregate analysis of risk is a Monte Carlo simulation performed on a portfolio of risks that an insurer or reinsurer holds. A portfolio can cover risks related to catastrophic events such as earthquakes or floods, and may comprise tens of thousands of contracts. The contracts generally follow an 'eXcess of Loss' (XL) [10], [11] structure providing coverage for single event occurrences or multiple event occurrences, or a combination of both single and multiple event occurrences. Each trial in the analysis simulation represents a view of the occurrence of catastrophic events and the order in which they occur within a contractual year and how they will interact with complex treaty terms to produce an aggregated loss. A pre-simulated Year Event Table (YET) containing between several thousand and millions of alternative views of a single contractual year is used as input. The output of aggregate analysis is a Year Loss Table (YLT). From a YLT, an insurer or a reinsurer can derive important portfolio risk metrics such as the Probable Maximum Loss (PML) [12], [13] and the Tail Value at Risk (TVAR) [14], [15] which are used for both internal risk management and reporting to regulators and rating agencies.

In this paper, a MapReduce analysis of portfolios of catastrophic risk is proposed and implemented using the Hadoop [16], [17], [18] platform. The algorithm must rapidly consume large amounts of data in the form of the YET and Event Loss Tables (ELT). Therefore, the challenge of organising input data efficiently and of applying parallelism within the algorithm are considered. The MapReduce model lends itself well towards solving embarrassingly parallel problems such as the aggregate analysis of risk, and is hence chosen to implement the algorithm. The algorithm employs two MapReduce rounds to perform both the numerical computations as well as to manage data efficiently. The algorithm is implemented on the Apache Hadoop platform. The preliminary results obtained from the experiments of the analysis indicate that the MapReduce model can be used to scale the analysis over multiple nodes of a cluster, and parallelism can be exploited in the analysis for faster numerical computations and data management.

The remainder of this paper is organised as follows. Section II considers the MapReduce algorithm for the analysis of aggregate risk. Section III presents the implementation of the MapReduce algorithm on the Apache Hadoop Platform and the preliminary results obtained experimental studies. Section IV concludes this paper by considering future work.

[1]Corresponding Author

II. MapReduce Algorithm for Analysis of Aggregate Risk

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 [16], [17], [18], 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 the analysis of aggregate risks. 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.

There are three inputs to the algorithm for the analysis of aggregate risk, 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).

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

Algorithm 1 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 2) 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 3). The output of $MapReduce_1$ is a Layer Loss Table LLT.

```
Input  : YET, ELT pool, PF
Output: YLT
1 forall the Programs of P do
2     forall the Layers L in P do
3         LLT ← MapReduce_1(L, YET)
4     end
5 end
6 YLT ← MapReduce_2(LLTs)
```

Algorithm 1: Pseudo-code for Parallel Analysis of Aggregate Risk

The second round of MapReduce jobs, denoted as $MapReduce_2$ are launched for aggregating all the $LLTs$ in each Program to a YLT.

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 2 and Algorithm 3 show how parallelism is achieved by using the Map and Reduce functions in a first round at the Layer level. Algorithm 2 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). The loss, l_E associated with an Event, E in the ELT is fetched from memory in line no. 2. 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 3 shows the Reduce function in the first MapReduce round. 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 $< T, l_T >$. 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.

Algorithm 4 and Algorithm 5 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. Algorithm 4 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 5 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, 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 : $< T, E >$
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 Financial Terms to l_E
4 | $l'_E \leftarrow l'_E$ + l_E
5 **end**
6 Emit(T, l'_E)

Algorithm 2: Pseudo-code for Map function in $MapReduce_1$ of the Analysis of Aggregate Risk

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 3: Pseudo-code for Reduce Function in $MapReduce_1$ of the Analysis of Aggregate Risk

III. Implementation and Experiments on the Hadoop Platform

The experimental platform for implementing the MapReduce algorithm 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 interconnected via Infiniband.

Input : $LLTs$
Output: $< T, l_T >$

1 **for** *each* T *in* LLT **do**
2 | Emit($< T, l_T >$)
3 **end**

Algorithm 4: Pseudo-code for Map function in $MapReduce_2$ of the Analysis of Aggregate Risk

Input : $< T, L_T >$
Output: $< T, l'_T >$

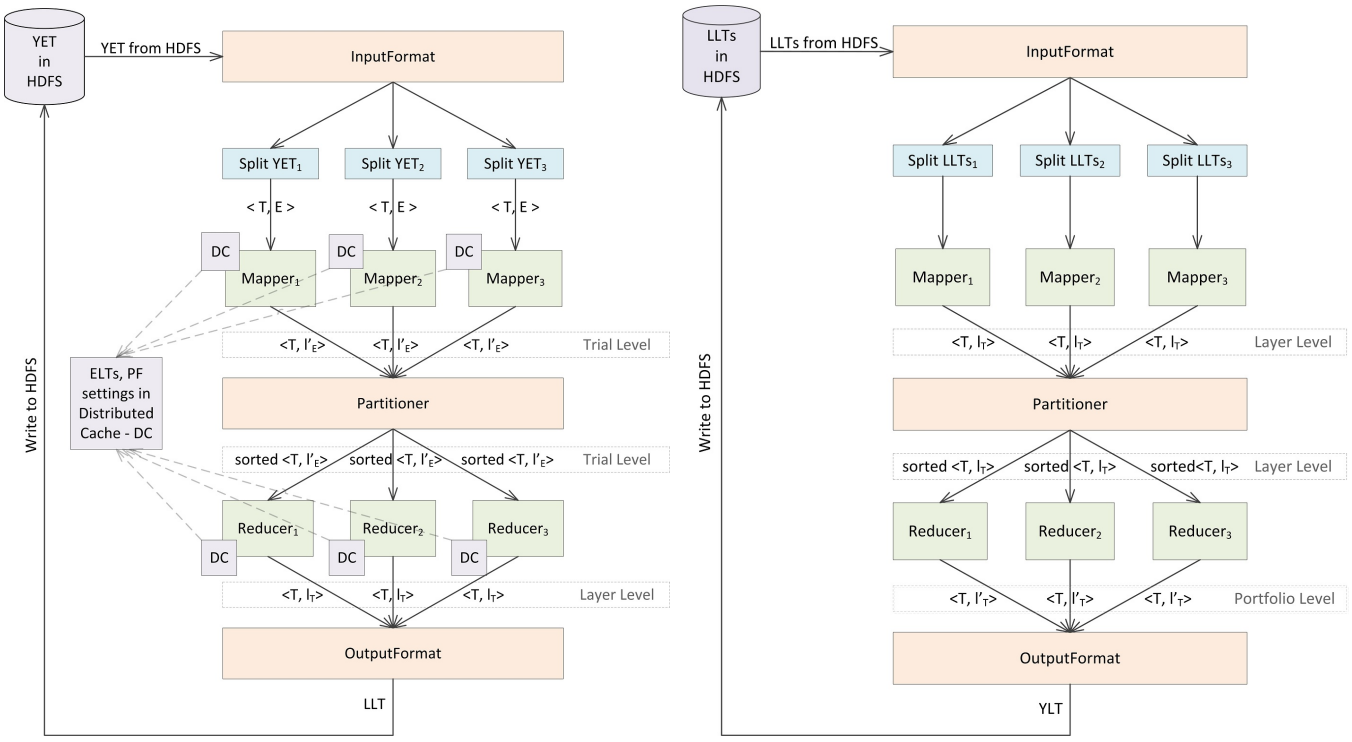
1 **for** *each* l_T *in* L_T **do**
2 | $l'_T \leftarrow l'_T + l_T$
3 **end**
4 Emit($< T, l'_T >$)

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

Apache Hadoop, an open-source software framework is used for implementing the MapReduce analysis of aggregate risk. Other available frameworks [19], [20] 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 1 to generate the Year Loss Table YLT which is again stored on the HDFS. The two



(a) First MapReduce round

(b) Second MapReduce round

Figure 1: MapReduce rounds in the Hadoop implementation

MapReduce rounds are illustrated in Figure 1.

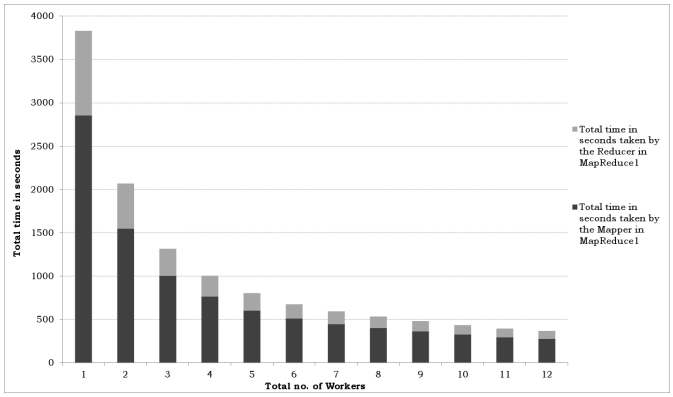
In the first MapReduce round the `InputFormat` interface splits the YET based on the number of Mappers specified for the MapReduce round. The Mappers are configured such that they also receive the $ELTs$ covered by one Layer which are contained in the distributed cache. The Mapper applies secondary uncertainty 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.

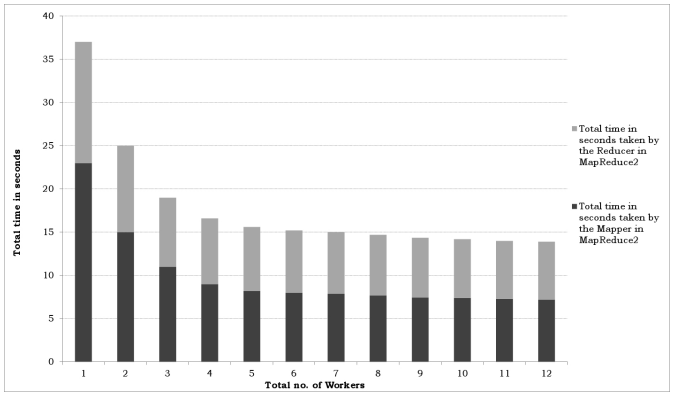
A. Results

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.

Figure 2 shows two bar graphs for the total time taken in seconds for the MapReduce rounds when the workers are varied between 1 and 12; Figure 2a for the first MapReduce round and Figure 2b for the second MapReduce round. In the first MapReduce round the best timing performance is achieved on 12 Mappers and 12 Reducers taking a total of 370 seconds, with 280 seconds for the Mapper and 90 seconds for the Reducer. Over 85% efficiency is achieved in each case using multiple worker nodes compared to 1 worker. This round is most efficient on 3 workers achiev-

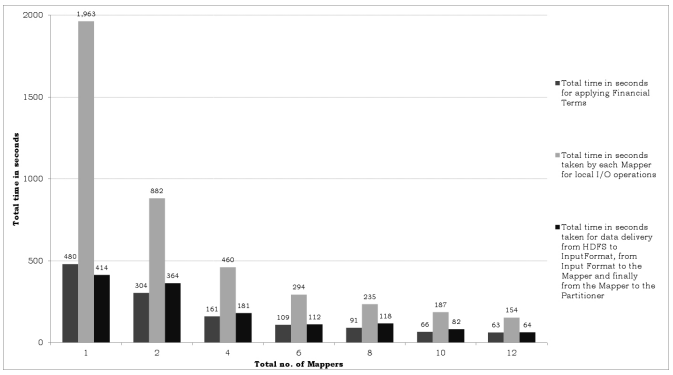


(a) First MapReduce round

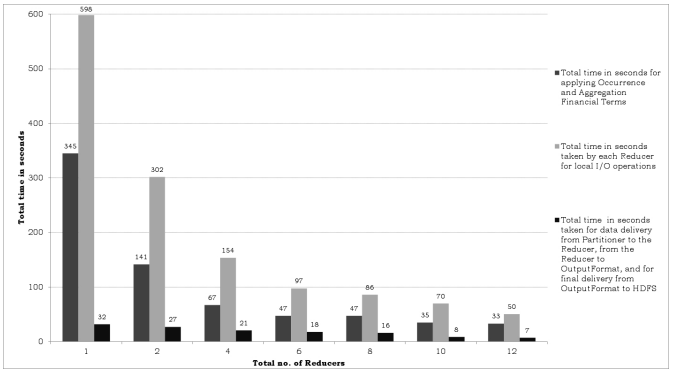


(b) Second MapReduce round

Figure 2: Number of workers vs total time taken in seconds for the MapReduce rounds in the Hadoop implementation



(a) No. of Mappers vs Time time taken for (a) applying Financial Terms, (b) local I/O operation by each Mapper, and (c) data delivery



(b) No. of Reducers vs Time time taken for (a) applying Occurrence and Aggregate Financial Terms, (b) local I/O operation by each Reducer, and (c) data delivery

Figure 3: Profiled time for the first MapReduce round in the Hadoop implementation

ing an efficiency of 97% and the performance deteriorates beyond the use of four workers on the cluster employed. In the second MapReduce round the best timing performance is achieved again on 12 Mapper and 12 Reducers taking a total of 13.9 seconds, with 7.2 seconds for the Mapper and 6.7 seconds for the Reducer. Using 2 workers has the best efficiency of 74%; the efficiency deteriorates beyond this. The second MapReduce round has performed poorly compared to the first round as there are large I/O and initialisation overheads on the workers.

Figure 3 shows two bar graphs in which the time for the first MapReduce round is profiled. For the Mapper the time taken into account is (a) for applying Financial Terms, (b) for local I/O operations, and (c) for data delivery from the HDFS to the InputFormat, from the InputFormat to the Mapper, and from the Mapper to the Partitioner. For the Reducer the time taken into account is (a) for applying Occurrence and Aggregate Financial Terms, (b) for local I/O operations, and (c) for data delivery from the Partitioner to the Reducer, from the Reducer to the OutputFormat and from the OuputFormat to HDFS. 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 $1/4^{th}$ 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 the numerical computations and for large data processing and delivery.

Figure 4 shows a bar graph for the time taken in seconds for the second MapReduce round on 12 workers when the number of Layers are varied from 1 to 5000. There is a steady increase in the time taken for data processing and delivery by the Mapper and the Reducer, and it seems that the time step starts to fall which will result in the flattening of the trend.

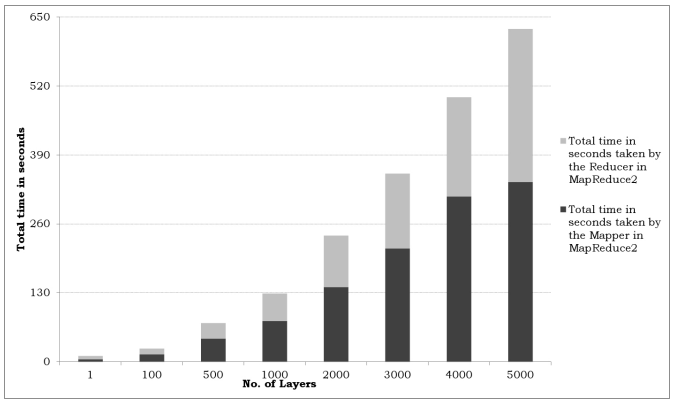


Figure 4: Number of Layers vs the total time in seconds for the second MapReduce round

The results indicate that there is scope for achieving high efficiency and speedup for numerical computations and large data processing and delivery within the Hadoop system. However, it is apparent that the large overhead for local I/O operations on the workers are restraining better performance. This large overhead is a resultant of the bottleneck in the connectivity between the worker nodes and the latency in processing data from local drives. Therefore, efforts need to be made towards reducing the I/O overhead to exploit the full benefit of the Hadoop MapReduce model.

IV. Conclusion

This paper has presented how the MapReduce model can meet the requirements of rapidly consuming large volumes of data for the analysis of portfolios of catastrophic risk. The challenge of handling large data and applying parallelism are handled by the Map and Reduce functions. An algorithm for the analysis of aggregate risk is proposed to incorporate the MapReduce model and implemented on the Apache Hadoop platform. The experimental results show the feasibility of MapReduce for parallel numerical computations and data management in the analysis.

Future work will be directed towards optimising the implementation for reducing the local I/O overheads to achieve better speedup. Efforts will be made towards incorporating additional financial filters, such as secondary uncertainty for in-depth analysis of aggregate risk.

References

[1] G. Connor, L. R. Goldberg and R. A. Korajczyk, “Portfolio Risk Analysis,” Princeton University Press, 2010.

[2] A. Melnikov, “Risk Analysis in Finance and Insurance, Second Edition” CRC Press, 2011.

[3] A. K. Bahl, O. Baltzer, A. Rau-Chaplin and B. Varghese, “Parallel Simulations for Analysing Portfolios of Catastrophic Event Risk,” Workshop of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2012.

[4] J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” Communications of the ACM, Vol. 51, No. 1, 2008, pp. 107-113.

[5] R. R. Anderson and W. Dong, “Pricing Catastrophe Reinsurance with Reinstatement Provisions Using a Catastrophe Model,” Casualty Actuarial Society Forum, Summer 1998, pp. 303-322.

[6] G. G. Meyers, F. L. Klinker and D. A. Lalonde, “The Aggregation and Correlation of Reinsurance Exposure,” Casualty Actuarial Society Forum, Spring 2003, pp. 69-152.

[7] W. Dong, H. Shah and F. Wong, “A Rational Approach to Pricing of Catastrophe Insurance,” Journal of Risk and Uncertainty, Vol. 12, 1996, pp. 201-218.

[8] K. -H. Lee, Y. -J. Lee, H. Choi, Y. D. Chung and B. Moon, “Parallel Data Processing with MapReduce: A Survey”, SIGMOD Record, Vol. 40, No. 4, 2011, pp. 11-20.

[9] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy and R. Sears, “MapReduce Online,” EECS Department, University of California, Berkeley, USA, Oct 2009, Technical Report No. UCB/EECS-2009-136.

[10] D. Cummins, C. Lewis and R. Phillips, “Pricing Excess-of-Loss Reinsurance Contracts Against Catastrophic Loss,” The Financing of Catastrophe Risk, Editor: K. A. Froot, University of Chicago Press, 1999, pp. 93-148.

[11] Y. -S. Lee, “The Mathematics of Excess of Loss Coverages and Retrospective Rating - A Graphical Approach,” Casualty Actuarial Society Forum, 1988, pp. 49-77.

[12] G. Woo, “Natural Catastrophe Probable Maximum Loss,” British Actuarial Journal, Vol. 8, 2002.

[13] M. E. Wilkinson, “Estimating Probable Maximum Loss with Order Statistics,” Casualty Actuarial Society Forum, 1982, pp. 195-209.

[14] A. A. Gaivoronski and G. Pflug, “Value-at-Risk in Portfolio Optimisation: Properties and Computational Approach,” Journal of Risk, Vol. 7, No. 2, 2005, pp. 1-31.

[15] P. Glasserman, P. Heidelberger and P. Shahabuddin, “Portfolio Value-at-Risk with Heavy-tailed Risk Factors,” Mathematical Finance, Vol. 12, No. 3, 2002, pp. 239-269.

[16] T. White, “Hadoop: The Definitive Guide,” 1st Edition, O’Reilly Media, Inc., 2009.

[17] Apache Hadoop Project: http://hadoop.apache.org/ [Last Accessed: 14/01/2013]

[18] K. Shvachko, K. Hairong, S. Radia and R. Chansler, “The Hadoop Distributed File System,” Proceedings of the 26th IEEE Symposium on Mass Storage Systems and Technologies, 2010, pp. 1-10.

[19] Amazon Elastic Map Reduce (EMR): http://aws.amazon.com/elasticmapreduce/ [Last accessed: 14/01/2013]

[20] Google MapReduce: https://developers.google.com/appengine/docs/python/dataprocessing/overview [Last accessed: 14/01/2013]