QuPARA: **Qu**ery-Driven Large-Scale **P**ortfolio **A**ggregate **R**isk **A**nalysis on MapReduce

A. Rau-Chaplin, B. Varghese, D. Wilson, Z. Yao, and N. Zeh
Risk Analytics Lab, Dalhousie University
Halifax, Nova Scotia, Canada
Email: {arc, varghese, yao, nzeh}@cs.dal.ca

***Abstract*—Modern insurance and reinsurance companies use stochastic simulation techniques for portfolio risk analysis. Their risk portfolios may consist of thousands of reinsurance contracts covering millions of individually insured locations. To quantify risk and to help ensure capital adequacy, each portfolio must be evaluated in up to a million simulation trials, each capturing a different possible sequence of catastrophic events (e.g., earthquakes, hurricanes, etc.) over the course of a contractual year.**

We present a flexible framework for portfolio risk analysis that can answer a rich variety of catastrophic risk queries. Rather than aggregating simulation data in order to produce a small set of high-level risk metrics efficiently (as done in production risk management systems), our focus is on queries on unaggregated or partially aggregated data. The goal is to allow analysts to obtain answers to a wide variety of unanticipated but natural ad hoc queries, which can help actuaries or underwriters to better understand the multiple dimensions (e.g., spatial correlation, seasonality, peril features, construction features, financial terms, etc.) that can impact portfolio risk and thus company solvency.

We implemented a prototype system, called QuPARA, using Apache's Hadoop implementation of the MapReduce paradigm. This allows the user to utilize large parallel compute servers in order to answer ad hoc queries efficiently even on very large data sets typically encountered in practice. We describe the design and implementation of QuPARA and present experimental results that demonstrate its feasibility.

Keywords*—*ad hoc risk analytics; aggregate risk analytics; portfolio risk; MapReduce; Hadoop

I. Introduction

Modern insurance/reinsurance companies use a stochastic simulation technique, called *aggregate analysis*, for portfolio risk analysis and pricing [1], [2], [3], [4]. A risk portfolio may consist of thousands of annual reinsurance contracts covering millions of individually insured locations. To quantify annual portfolio risk, each portfolio must be evaluated in up to a million simulation trials, each consisting of a sequence of possibility thousands of catastrophic events, such as earthquakes, hurricanes or floods. Each trial captures one scenario how globally distributed catastrophic events may unfold over the course of a year.

Aggregate analysis is computationally intensive and data-intensive. Production analytical systems exploit parallelism and ruthlessly aggregate results in order to efficiently produce a small set of standard portfolio risk metrics of interest to regulatory bodies, rating agencies, and an organization's risk management team, such as *probable maximum loss* (PML) [5], [6] and *tail value-at-risk* (TVaR) [7], [8]. While these systems can efficiently aggregate terabytes of simulation results into a small set of key metrics, they are typically not suitable for answering ad hoc queries that can help actuaries or underwriters to better understand the multiple dimensions of risk that can impact a portfolio, such as spatial correlation, seasonality, peril features, construction features, and financial terms.

In this paper, we propose a framework for aggregate risk analysis that allows users with extensive mathematical and statistical skills but perhaps limited programming background, such as risk analysts, to pose a rich variety of complex ad hoc risk queries. The user formulates their query by defining SQL-like filters. The framework then answers the query based on these filters, without requiring the user to make changes to or even have knowledge of the core implementation of the framework or to reorganize the input data of the analysis. The challenges that arise due to the amounts of data to be processed and due to the complexity of the required computations are largely hidden from the user.

Our prototype implementation of this framework for Query-Driven Large-Scale Portfolio Aggregate Risk Analysis (QuPARA) uses Apache's Hadoop [9], [10] implementation of the MapReduce programming model [11], [12], [13] to exploit parallelism, and Apache Hive [14], [15] to support ad hoc queries. Even though QuPARA is not as fast as a production system on the narrow set of standard portfolio metrics, it supports a wide variety of ad hoc queries and can answer them efficiently. For example, our experiments demonstrate that an industry-size risk analysis with 1,000,000 simulation trials, 1,000 events per trial, and on a portfolio consisting of 3,200 risk transfer contracts (layers) with an average of 5 event loss tables per layer can be carried out on a 16-node Hadoop cluster in just over 20 minutes.

The remainder of this paper is organized as follows. Section II gives an overview of reinsurance risk analysis. Section III proposes our risk analysis framework and describes its implementation. Section IV presents a performance evaluation of our framework. Section V presents conclusions.

II. An Overview of Risk Analysis

A reinsurance company typically holds a *portfolio* of programs that insure primary insurance companies against large-scale losses associated with catastrophic events. Each *program* contains data that describes (1) the buildings to be insured (the *exposure*), (2) the modelled risk to that exposure (the *event loss tables*), and (3) a set of risk transfer contracts (the *layers*).

978-1-4799-1293-3/13/$31.00 ©2013 IEEE

The *exposure* is represented by a table, one row per building covered, that lists the building's location, construction details, primary insurance coverage, and replacement value. The modelled risk is represented by an *event loss table* (ELT). This table lists for each of a large set of possible catastrophic events the expected loss that would occur to the exposure should the event occur and is produced using stochastic region peril models [16]. Finally, each *layer* (risk transfer contract) is described by a set of financial terms that includes aggregate deductibles and limits (i.e., deductibles and maximal payouts to be applied to the sum of losses over the year) and per-occurrence deductibles and limits (i.e., deductibles and maximal payouts to be applied to each loss in a year), plus other financial terms.

As an example, consider a Japanese earthquake program. The exposure might list 2 million buildings, including their locations, construction details, primary insurance terms, and replacement values. The ELT might, for each of 100,000 possible earthquake events, give the sum of the losses expected to the exposure should the event occur. A risk transfer contract might consist of two layers, one a per-occurrence layer that pays out 60% of the losses between 160 million and 210 million associated with a single catastrophic event, the other an aggregate layer covering 30% of the losses between 40 million and 90 million that accumulate due to earthquakes over the course of a year.

Given a reinsurance company's portfolio, the most fundamental type of query computes an Exceedance Probability (EP) curve, which represents, for each of a set of user-specified loss values, the probability that the total claims a reinsurer will have to pay out exceeds this value. There is no computationally feasible closed-form expression for computing such an EP curve over hundreds of thousands of events and millions of individual exposures. Consequently a simulation approach must be taken. The idea is to perform a stochastic simulation based on a *year event table* (YET). This table describes a large number of trials, each representing one possible sequence of catastrophic events that might occur in a given year. This YET is generated by an event simulator whose details are beyond the scope of this paper. Our focus is on computing the expected loss distribution (i.e., EP curve) for a given portfolio, given a particular YET. Given the sequence of events in a trial, the loss for this particular trial can be computed, and the overall loss distribution is obtained from the losses computed for the set of trials.

While computing the EP curve for the company's entire portfolio is critical in assessing a company's solvency, analysts are often interested in answering more detailed queries with the goal of analyzing such things as cash flow throughout the year, diversity of the portfolio, financial impact of adding a new contract or contracts to the portfolio, and many others. The following is a representative, but far from complete, list of queries analysts may be interested in.

EP Curves with secondary uncertainty: Basic aggregate risk analysis addresses the *primary uncertainty* whether an event will occur and associates a loss value with each event should it occur. However, even if an event occurs, the loss value may vary based on *secondary uncertainty* arising due to unknown parameters of the exposure or hazard and their interactions. This secondary uncertainty is captured by associating a probability distribution of loss values rather than a single loss value with each event. Taking secondary uncertainty into account when computing exceedance probabilities is computationally intensive due to the statistical tools employed but is essential in many applications.

Return period losses (RPL) by line of business (LOB), class of business (COB) or type of participation (TOP): A layer defines coverage on different types of exposures and the type of participation. Exposures can be classified by class of business (COB) or line of business (LOB) (e.g., marine, property or engineering coverage). The way in which the contractual coverage participates when a catastrophic event occurs is defined by the type of participation (TOP). Decision makers may want to know the loss distribution of a specific layer type in their portfolios, which requires the analysis to be restricted to layers covering a particular LOB, COB or TOP.

Region/peril losses: This type of query calculates the expected losses or a loss distribution for a set of geographic regions (e.g., Florida or Japan), a set of perils (e.g., hurricane or earthquake) or a combination of region and peril. This allows the reinsurer to understand both what types of catastrophes provide the most risk to their portfolio and in which regions of the globe they are most heavily exposed to these risks.

Multi-marginal analysis: Given the current portfolio and a small set of potential new contracts, a reinsurer has to decide which contracts to add to the portfolio. Adding a new contract means additional cash flow but also increases the exposure to risk. To help with the decision which contracts to add, multi-marginal analysis calculates the difference between the loss distributions for the current portfolio and for the portfolio with any subset of these new contracts added.

Stochastic exceedance probability (StEP) analysis: After a natural disaster not in their event catalogues occurs, catastrophe modelling [16] vendors try to estimate the distribution of possible loss outcomes for that event, in order to include the event in their catalogues. One way of doing this is to find a weighted combination of the distributions of existing events in the catalogue that best represents the actual occurrence. Given the list of events and their associated weights, the combined distribution is computed using a customized YET with one event per trial. This is event is chosen from the set of possible events; each event is chosen with a probability proportional to its weight. A standard aggregate analysis using this YET produces a loss distribution of the new event, including various statistics, such as mean, variance, and quantiles.

Periodic loss distribution: Many natural catastrophes have a seasonal component, that is, do not occur uniformly throughout the year. For example, hurricanes on the Atlantic coast occur between July and November. As a result, the reinsurer may be interested in how their potential losses fluctuate throughout the year, for example to reduce their exposure through reduced contracts or increased retrocessional coverage during riskier periods. To aid in these decisions, a periodic loss distribution represents the loss distribution for different periods of the year, such as quarters, months or weeks.

III. QuPARA Framework

In this section, we present our QuRARA framework. First we give an overview of the steps involved in answering an aggregate query sequentially. QuPARA provides a parallel version of this sequential algorithm.

Every query is answered from the portfolio and the YET in two phases. Phase 1 computes a *year loss table* (YLT) listing the losses incurred by the events in each trial. Phase 2 computes the query answer from the YLT.

Phase 1 is the computationally intensive part of the algorithm. It consists of five nested loops iterating over the trials, the events in each trial, the programs in the portfolio, the layers in each program, and the ELTs covered by each layer. The innermost loop looks up the loss associated with the current event X in the current ELT and aggregates these losses in the current layer's *layer loss* l_L. Each such layer loss l_L is added to the current program's *program loss* l_P for event X after applying the layer's per-occurrence and aggregate financial terms to l_L. The program losses are aggregated into event X's *portfolio loss* l_{PF}. The portfolio losses for all events are aggregated to yield the total *trial loss* l_T of the entire trial.

Depending on the query, the YLT is populated with loss values at different levels of aggregation. At one extreme, if only the loss distribution for the entire portfolio is of interest, there is one loss value per trial. At the other extreme, if the filtering of losses based on some combination of region, peril, LOB, etc. is required, the YLT contains one loss value for each ⟨trial, program, layer, ELT, event⟩ tuple.

A. Parallelization Using MapReduce

The division of the computation of a query answer into first computing a YLT and then computing the query answer from the YLT leads to a natural MapReduce-based parallelization strategy employed by QuPARA. Since the iterations of the outer loop of the YLT computation for different trials are independent of each other, the computation of the YLT can be carried out by independent mappers, one per trial. The computation of the query answer is done by one or more reducers.

Figure 1 visualizes the design of QuPARA. The framework is split into a front-end offering a *query interface* to the user, and a back-end consisting of a *distributed file system* and a *query engine* that executes the query using MapReduce. Next we describe these three components in detail, starting with the distributed file system because the description of the tables it stores is necessary to understand the query interface and implementation of the query engine.

B. Distributed File System and Intermediate Tables

The distributed file system, implemented using Hadoop's HDFS [17], stores the portfolio and YET in a number of tables:

- The *year event table YET* stores tuples ⟨*trial_ID*, *event_ID*, *time_Index*, *z_PE*⟩. *trial_ID* is a unique trial identifier. *event_ID* is a unique identifier of an event in the event catalogue. *time_Index* determines the position of the event occurrence in the sequence of events in the trial. *z_PE* is a random number specific to the program and event occurrence.
- The *layer table LT* stores tuples ⟨*layer_ID*, *cob*, *lob*, *tob*, *elt_IDs*⟩. *layer_ID* is a unique layer identifier. *cob* is an industry classification of perils insured and the related exposure. Thus, it groups homogeneous risks. *lob* defines a set of one or more related products or services where a business generates revenue. *tob* describes how reinsurance coverage and premium payments are calculated. *elt_IDs* is a list of ELT IDs that are covered by the layer.
- The *layer list table LLT* contains tuples ⟨*layer_ID*, *occ_Ret*, *occ_Lim*, *agg_Ret*, *agg_Lim*⟩. Each entry represents the layer identified by *layer_ID*. *occ_Ret* is the occurrence retention (deductible) for an individual occurrence loss. *occ_Lim* is the occurrence limit (coverage) for occurrence losses in excess of the occurrence retention. *agg_Ret* is the aggregate retention (deductible) for the annual cumulative loss. *agg_Lim* is the aggregate limit (coverage) for annual cumulative losses in excess of the aggregate retention.
- The *event loss table pool ELTP* contains tuples ⟨*elt_ID*, *region*, *peril*⟩. Each such entry associates a particular type of peril and a particular region with the ELT with ID *elt_ID*.
- The *extended event loss table EELT* contains tuples ⟨*event_ID*, *z_E*, *mean_Loss*, *sigma_I*, *sigma_C*, *max_Loss*⟩. *event_ID* is the unique identifier of an event in the event catalogue. *z_E* is a random number specific to the event occurrence. *mean_Loss* and *max_Loss* denote the mean and maximum expected loss incurred if the event occurs, respectively. *sigma_I* represents the variance of the loss distribution for this event. *sigma_C* represents the error of the event-occurrence dependencies.
- The *event catalogue ECT* contains tuples ⟨*event_ID*, *region*, *peril*⟩ associating a region and type of peril with each event.

In addition to these tables stored in the distributed file system and representing the data used to answer a query, the back-end manipulates the following temporary tables that hold intermediate results produced as part of the query answering process.

- The *combined event loss table CELT* is constructed by each mapper in memory from the ELTs corresponding to the user's query. It associates with each event ID *event_ID* a list of tuples ⟨*elt_ID*, *z_E*, *mean_Loss*, *sigma_I*, *sigma_C*, *max_Loss*⟩, which is the loss information for event *event_ID* stored in the extended ELT with ID *elt_ID*.
- The *year event loss table YELT* is produced by the mapper for consumption by the combiner. It contains tuples ⟨*trial_ID*, *event_ID*, *time_Index*, *estimated_Loss*⟩.
- The *year region peril loss table YRPLT* contains tuples ⟨*trial_ID*, *time_Index*, *region*, *peril*, *estimated_Loss*⟩, listing for each trial the estimated loss at a given time, in a given region, and due to a particular type of peril. This table is produced by the combiner for consumption by the reducer.

C. Query Interface

The query interface offers a web-based portal through which the user can issue ad hoc queries in an SQL-like syntax. The query is then parsed and passed to the query engine. Queries are expressed by specifying three filters and two aggregation queries. The filters allow the user to focus their queries on specific geographic regions, types of peril, etc. by selecting the appropriate entries from the data tables they operate on. The aggregation queries specify how the data produced by the mappers is to be aggregated to produce the final query answer. We illustrate these five query components using an example query that aims to generate a report on seasonal loss value-at-risk (VaR) with confidence level of 99% due to hurricanes and floods that affect commercial properties in Florida. The query

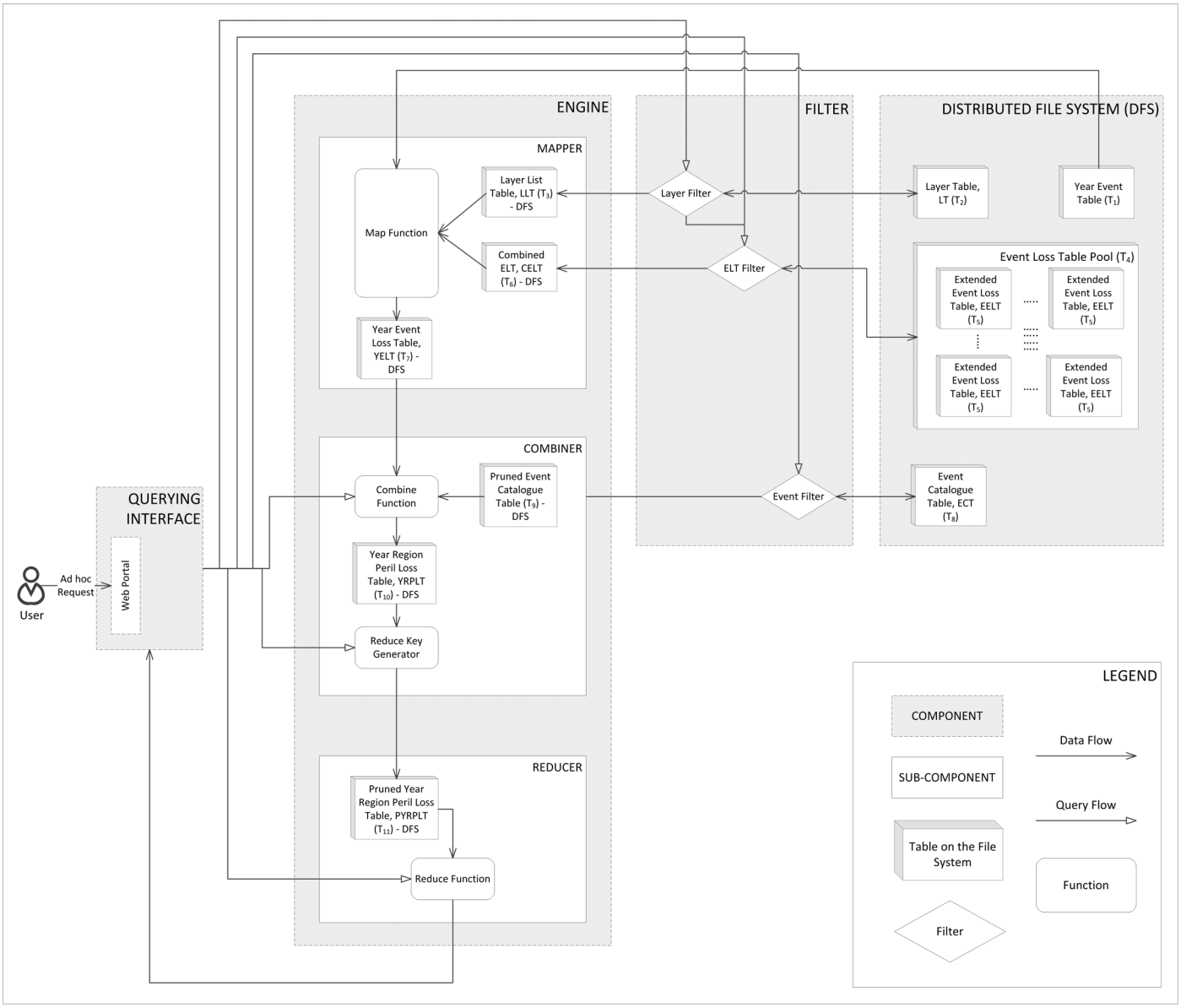

Fig. 1. The Query-Driven Portfolio Aggregate Risk Analysis (QuPARA) framework

engine employs Apache Hive [14], built on top of HDFS, to implement these filters and aggregation queries.

Q1. Layer filter: The *layer filter* extracts a set of layers from the layer table *LT* and passes this list of layers to the mapper as the "portfolio" to be analyzed by the query. The list of selected layers is also passed to the ELT filter for selection of the relevant ELTs. In our example query, we are interested in all layers covering commercial properties, which translates into the following SQL query:

SELECT * **FROM** *LT*
WHERE *lob* **IN** *commercial*

Q2. ELT filter: The *ELT filter* is used to select, from the ELT pool *ELTP*, the set of ELTs required by the layer filter. In our example query, we are interested in all ELTs covered by layers returned by the layer filter and which cover Florida (FL) as the region and hurricanes (HU) and floods (FLD) as perils:

SELECT *elt_ID* **FROM** *ELTP*
WHERE *elt_ID* **IN** *Q1*
AND *region* **IN** *FL*
AND *peril* **IN** *HU*, *FLD*

Q3. Event filter: The *event filter* selects features of events from the event catalogue *ECT* required to group estimated losses in the YELT. In our example query, this filter is not needed, but if the ELT filter covered multiple regions and the query required grouping by region, we would require the region of each event, which can be retrieved using the following query:

SELECT *event_ID* **AND** *region* **FROM** *ECT*

Q4. Per-trial aggregation: The *per-trial aggregation* query

instructs the combiner to aggregate the entries in the YELT for production of the final query output. In our example query, we are interested in grouping the events in a trial by season and aggregating the losses of all events in each season:

SELECT *trial_ID*, SUM(*estimated_Loss*) **FROM** *YELT*
GROUP BY SEASON(*time_Index*)

Q5. Output generation: This query is provided to the reducer to define the final output of the user request. In our example query, the seasonal loss Value-at-Risk (VaR) with 99% confidence level is generated using the following query:

SELECT *VaR* **IN** 0.01 **FROM** *YRPLT*

D. Query Engine

The query engine is implemented using Apache Hadoop [9], [10], [18] and employs a singe MapReduce round to evaluate the query. The input of each mapper consists of a part of the YET covering a subset of trials. Conceptually, there is one mapper per trial in the YET, but we use a partition of the YET into groups of trials to reduce the overhead of starting many mappers. The other tables needed by the mapper are made available to the mapper using HDFS's *distributed cache* feature, which allows small data files to be shared between the nodes of a cluster. Each mapper constructs a portion of the YLT from its portion of the YET. The computation of each mapper is identical to an iteration of the outer loop of the sequential algorithm. The YLT portion of each mapper is passed to a combiner associated with it for pre-aggregation of the loss information in this YLT portion. (A combiner is an extension of the MapReduce framework available in Hadoop. It allows reducer-like aggregation of the data produced by a single mapper, in order to reduce the amount of data to be processed in the shuffle step.) The combiners send their output to one or more reducers, which produce the final query output. For most queries, whose output consists of a single loss distribution, there is only one reducer. Multi-marginal analysis is an example where multiple loss distributions are computed, one per subset of potential contracts to be added to the portfolio. In this case, we have one reducer for each subset, and each reducer produces the loss distribution for its corresponding subset of contracts.

1) Mapper: Each mapper retrieves the set of ELTs required for the query from the distributed cache according to the layer and ELT filters specified in the query. The ELT filter produces the set of IDs of the requires EELTs. The mapper retrieves the corresponding EELTs from the distributed cache and constructs a *combined ELT* associating a loss with each ⟨event, ELT⟩ pair in memory. The CELT speeds up the lookup of events in the relevant ELTs because only one lookup per event is required in the CELT as opposed to many lookups across multiple ELTs. For each trial in its portion of the YET, the mapper iterates over the sequence of events in this trial, looks up the ELTs recording non-zero losses for each event, and generates the corresponding ⟨trial, program, layer, ELT, event, loss⟩ tuple in the YLT, taking the layer's financial terms into account.

2) Combiner: The combiner executes the event filter to annotate each event with properties from the event catalogue needed for grouping and pre-aggregation. The event catalogue is stored in the distributed cache. It then aggregates the loss values for each trial using the per-trial aggregation query. If the output of the query consists of a single loss distribution, the loss values for each trial are aggregated into a single loss value for this trial. If, for example, a weekly loss distribution is required, the combiner aggregates the losses corresponding to the events in each week and sends each week's aggregate to a different reducer. Each reducer is then responsible for computing the loss distribution for one particular week.

3) Reducer: Each reducer receives one loss value per trial from the combiners. It executes the output generation query on the set of received loss values to produce the final output.

4) Optimization: Each mapper needs to hold the CELT it constructs in memory, which is not possible if the CELT is large. Similarly, the construction of a large CELT is costly but is effectively not parallelized because each mapper constructs its own copy of the CELT. The size of the CELT to be constructed depends directly on the number of layers covered by a query. Thus, to overcome the memory limitation caused by a large CELT and to parallelize the CELT construction, our implementation deviates from the above description of the QuPARA framework and splits it into two MapReduce rounds.

In the first round, we split the set of layers covered by a query into batches of 200 layers. On our hardware, a CELT covering 200 layers fit in memory and did not take long to construct. For each batch, we create a separate MapReduce job and allocate an equal number of nodes to each job. The mappers and combiners of this job implement the mapper part of QuPARA on their subset of layers as described in this section. The reducers, on the other hand, cannot compute the final query answer without knowledge of the results produced by the mappers in other jobs. Thus, each reducer of the first round simply outputs the data it receives, tagging it with a key identifying the portion of the query answer it would have been responsible to produce.

In the second round, the mappers read the data produced by the reducers of the first round and use the tags produced by thees reducers as keys to identify the reducer each input tuple needs to be sent to. No other processing of the input data is performed by the mappers. Thus, the reducers of the second round receive the same data as in a one-round implementation of QuPARA and can now produce the final query answer as described in this section.

The above strategy to parallelize the CELT construction and reduce the size of the portion of the CELT stored by each mapper represents a trade-off. A larger number of jobs in the first round reduces the size of the CELT portion to be computed and stored by each mapper and increases the parallelism in constructing the CELT. However, the mappers of each job need to read the entire YET to produce the YLT. Thus, a larger number of jobs also increases the number of times the YET is read, which is a substantial increase in I/O volume because the YET is the largest table processed to answer a risk analysis query. We determined experimentally that reducing the number of layers per job below 200 layers (and thus increasing the number of jobs) resulted only in small gains from parallelizing the CELT construction further that were outweighed by the increase in I/O volume for reading the YET. This is why we fixed the number of layers per job at 200 layers.

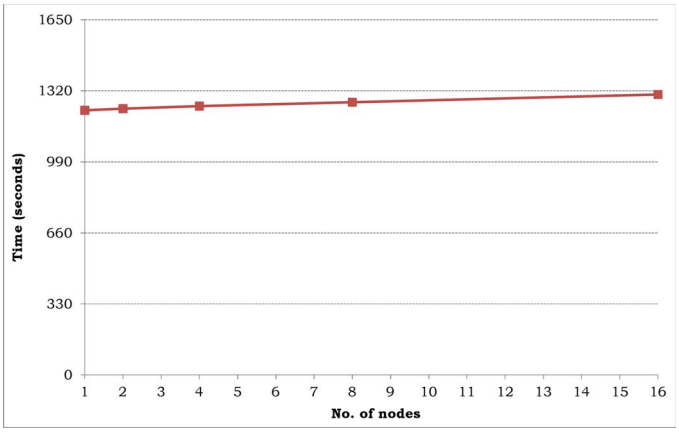

Fig. 2. Running time of QuPARA on an increasing number of nodes with a fixed number of layers per job and one job per node

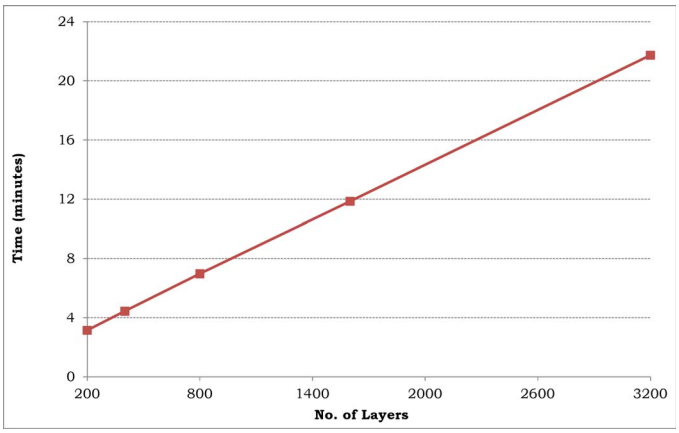

Fig. 3. Running time of QuPARA on an increasing number of layers with a fixed number of nodes

IV. Performance Evaluation

We evaluated QuPARA on a 16-node cluster connected via Gigabit Ethernet and with 306TB of global storage provided by a Sun SAM-QFS, a hierarchical storage system using RAID 5. Each node was an SGI C2112-4G3 with four quad-core AMD Opteron 8384 (2.7 GHz) processors, 64 GB of RAM, 500GB of local storage, and running Red Hat Enterprise Linux 4.8. The Java Virtual Machine version was 1.6. The Apache Hadoop version was 1.0.4. The Apache Hive version was 0.10.0.

Figure 2 shows the running time of an aggregate risk analysis query on between 1 and 16 nodes with 200 layers per node and with each layer covering 5 unique ELTs. Thus, between 1,000 and 16,000 ELTs were considered and the amount of computation per node was fixed. The YET contained 1,000,000 trials, each consisting of 1,000 events. The graph shows a very slow increase in the running time, which is due to the increase in the setup time required by the Hadoop job scheduler and the increase in network traffic (reflected in an increase in the time taken by the reducer) resulting from the higher number of nodes. This scheduling and network traffic overhead amounted to only 1.67%–7.13% of the total computation time, that is, if the hardware scales with the input size, QuPARA's processing time of a query remains nearly constant.

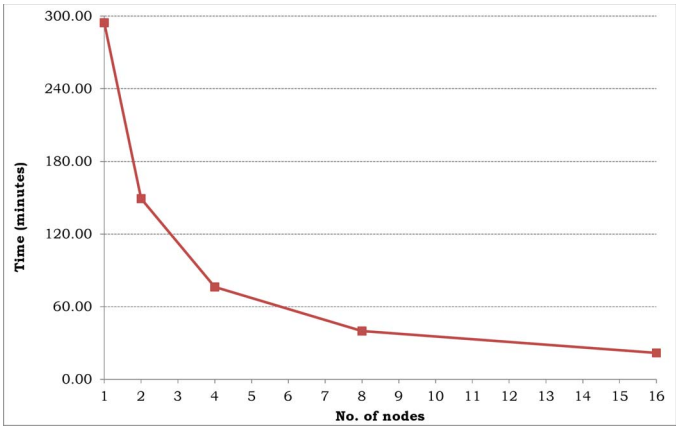

Fig. 4. Running time of QuPARA on a fixed input size using between 1 and 16 nodes

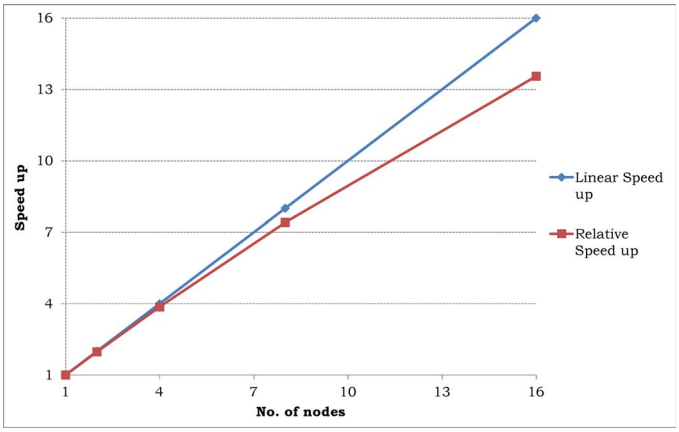

Fig. 5. Speed-up achieved on the QuPARA framework

Figure 3 shows the running time of a query on between 200 and 3200 layers using 16 nodes. Once again, each layer covered 5 ELTs, the YET contained 1,000,000 trials, each consisting of 1,000 events. The running time of QuPARA increased linearly with the input size as the number of layers increased. With the increase in the number of layers, the time taken for setup, I/O time, and the time for all numerical computations scale in a linear fashion. The time taken for building the combined ELT, by the reducer, and for clean up are constant. For 200 Layers, only 30% of the time was spent on computation; the remaining time accounts for system and I/O overheads. For 3200 layers, more than 80% of the time was spent on computation.

Figure 4 shows the running time of a query on 3,200 layers using between 1 and 16 nodes. Figure 5 shows the relative speed-up achieved in this experiment, that is, the ratio between the running time achieved on a single node and the running time achieved on up to 16 nodes. Up to 4 nodes, the speed-up was almost linear. Beyond 4 nodes, the speed-up started to decrease. At this point, the amount of computation to be parallelized was insufficient to mask the overhead of setting up the MapReduce computation, so the constant setup overhead amounted to a greater portion of the total running time.

Our experiments show that QuPARA is capable of answering ad-hoc aggregate risk analysis queries on industry-size data sets in a matter of minutes and scales well as input size and

the number of available nodes increases. Thus, it is a viable tool for analysts to carry out such analyses interactively.

V. Conclusions

Typical aggregate risk analysis systems performing aggregate risk analysis used in the industry are efficient for generating a small set of key portfolio metrics, such as PML or TVAR, but are not suitable for answering ad hoc queries useful to analysts and decision makers. In this paper, we presented an implementation of a framework for answering such queries efficiently using Apache Hadoop's MapReduce implementation and Apache's Hive query language. Our experiments demonstrate the feasibility of answering ad hoc queries on industry-size data sets efficiently using our framework and thus demonstrate its potential usefulness to analysts in the reinsurance industry.

References

[1] R. R. Anderson and D. W., "Pricing catastrophe reinsurance with reinstatement provisions using a catastrophe model," *Casualty Actuarial Society Forum*, pp. 303–322, Summer 1998.

[2] G. G. Meyers, F. L. Klinker, and D. A. Lalonde, "The aggregation and correlation of reinsurance exposure," *Casualty Actuarial Society Forum*, pp. 69–152, Spring 2003.

[3] W. Dong, H. Shaw, and F. Wong, "A rational approach to pricing of catastrophe insurance," *Journal of Risk and Uncertainty*, vol. 12, pp. 201–218, 1996.

[4] R. M. Berens, "Reinsurance contracts with a multi-year aggregate limit," *Casualty Actuarial Society Forum*, pp. 289–308, Spring 1997.

[5] G. Woo, "Natural catastrophe probable maximum loss," *British Actuarial Journal*, vol. 8, 2002.

[6] M. E. Wilkinson, "Estimating probable maximum loss with order statistics," *Casualty Actuarial Society Forum*, pp. 195–209, 1982.

[7] A. A. Gaivoronski and G. Pflug, "Value-at-risk in portfolio optimization: Properties and computational approach," *Journal of Risk*, vol. 7, no. 2, pp. 1–31, 2004-05.

[8] P. Glasserman, P. Heidelberger, and P. Shahabuddin, "Portfolio value-at-risk with heavy-tailed risk factors," *Mathematical Finance*, vol. 12, no. 3, pp. 239–269, 2002.

[9] T. White, *Hadoop: The Definitive Guide*, 1st ed. O'Reilly Media, 2009.

[10] Apache Hadoop project website. Last accessed: 25 May, 2013. [Online]. Available: http://hadoop.apache.org

[11] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[12] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears, "MapReduce online," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-136, October 2009, last accessed: 25 May, 2013. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-136.pdf

[13] 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, pp. 11–20, 2011.

[14] HiveQL website. Last accessed: 25 May, 2013. [Online]. Available: http://hive.apache.org

[15] E. Capriolo, D. Wampler, and J. Rutherglen, *Programming Hive*, 1st ed. O'Reilly Media, 2012.

[16] P. Grossi and H. Kunreuter, *Catastrophe Modelling: A New Approach to Managing Risk*. Springer, 2005.

[17] Hadoop distributed file system website. Last accessed: 25 May, 2013. [Online]. Available: http://hadoop.apache.org/docs/r1.0.4/hdfs_design.htmlS

[18] K. Shvachko, K. Hairong, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Proceedings of the 26th IEEE Symposium on Mass Storage Systems and Technologies*, 2010, pp. 1–10.