Scalable Data Analytics Platform for Enterprise Backup Management

Yang Song¹, Ramani Routray¹, and Yangyang Hou¹,²
¹IBM Almaden Research Center, San Jose, CA
²Computer Science Department, Purdue University, West Lafayette, IN
Email: yangsong, routrayr, yhou@us.ibm.com

Abstract—Data protection using backup is one of the most critical IT management operations to ensure business continuity, which is also constantly evolving due to the emerging challenges in the “Big Data Era.” In this paper, we introduce our ongoing research effort in designing intelligent enterprise backup management solutions by obtaining actionable insights from voluminous backup job metadata across data centers. Our proof-of-concept scalable data analytics framework integrated with enterprise backup architecture is introduced, and use case studies of leveraging big data analytics on enterprise backup framework are discussed.

I. INTRODUCTION

As one of the most traditional and common IT management tasks in data centers, data protection using backup is becoming more crucial for enterprise business operations as the criticality of data arises. Nevertheless, traditional backup solutions are criticized for slow processing time, unfavorable management complexity, expensive cost, and unpredictable performance in the “Big Data Era,” which are particularly exacerbated by the skyrocketing speed of data growth. For example, there are approximately 48 hours of video being uploaded to YouTube every minute, and the voluminous user-generated data must be protected using backup solutions with minimum human intervention and low cost. From a business perspective, advanced backup and recovery solutions which can better satisfy the more stringent business continuity requirements are strongly desired. By 2016, one third of the organizations is expected to change backup vendors due to the dissatisfaction of cost, management complexity, and limited capability [1]. It is also projected that the enterprise distributed system backup/recovery software market will grow to 5.2 billion by 2016, from 3.8 billion in 2011, which shows a significant growth rate. Therefore, designing reliable, easy-to-use, and intelligent backup solutions is of paramount importance to IT management teams to advance their business competitiveness and market leadership.

In this work, we introduce our on-going research focus of designing intelligent enterprise backup management solutions by mining actionable insights from historical backup job metadata across multiple data centers. In the rest of the paper, we will first introduce the enterprise backup architecture and the integration with our backup data analysis framework. Next, a proof-of-concept platform for scalable data mining which facilitates parallel implementations of data mining algorithms on massive data sets is presented. Several use case studies will be discussed to illustrate our research work on exploiting data mining techniques for efficient enterprise backup management with the aid of big data analytics techniques.

II. ENTERPRISE BACKUP MANAGEMENT AS A BIG DATA ANALYTICS PROBLEM

A. Enterprise Backup Architecture

As shown in Figure 1, a common enterprise backup architecture consists of multiple backup clients, backup servers, and storage backend devices. The backup clients have software backup/restore agents installed, e.g., IBM Tivoli Storage Manager [2], which are configured to communicate with one dedicated backup server for all backup and restore tasks. Each client usually specifies a backup policy, e.g., backup frequency (daily or hourly), backup type (full, incremental, or differential), a backup window (e.g., 3am-5am every day), or a combination of these specifications. For example, in Figure 1, backup client 1 is configured to communicate with backup Server A, which also manages the backup/recovery tasks for client 2 and 3, respectively. Client 1 can specify that “back up all new files on the C: drive of the guest OS between 3am to 5am every day.” Backup servers are connected to storage backend where the actual user data will be stored and replicated. During the backup window specified by the client, the corresponding backup server will contact the software agent and initiate the data transfer. It can also be configured...
in a policy that in case of a failed backup job, certain number of retries must be executed automatically. Depending on the backup type, e.g., full, incremental, and differential, the content that will be transferred from a client to its backup server may vary significantly in terms of data transfer size and backup duration. For the ease of management, backup clients are usually categorized into logical groups, a.k.a., zones, such as “Finance” and “Human Resources.” For each individual backup job, the backup server will record many pieces of information of the job, i.e., metadata, such as the associated client name, the associated zone name, the backup start time, the backup finish time, the size of data transfer, the experienced network throughput, the number of backup files, the outcome of the backup job (fail, success, or partial), the backup job type (full, incremental), and the number of open files encountered (opened by other applications and thus cannot be backed up), among many others. Such information is usually stored in an enterprise database server, e.g., DB2 or SQLServer. Note that our backup data analytics is performed on these backup job metadata rather than the actual user data from clients.

B. Big Data Analytics Platform for Backup Management

In order to exploit data mining techniques on collected backup job metadata, we integrate a big data analytics platform with the existing enterprise backup architecture, as shown in Figure 1. Our platform harnesses critical configuration and operational information from software and hardware components across all backup servers and analyzes the historical information to provide multi-tiered reporting, monitoring, alerting capabilities. Data from other management silos besides backup are also collected to bridge the information gap and create a single viewing, governance, and management console for the backup administrator. A unified management stack is enabled, which continuously collects each individual backup job’s attributes (more than 20 in the default setting), such as the client name, the start time, the finish time, the size of data transfer, the experienced network throughput, and the number of open files, along with their associated timestamps. The data set we are collecting consists of backup jobs for over half million of clients, over 2000 servers, and over 3000 zones in total. Our historical backup jobs metadata database contains up to 3 years of records for all client, server, and zone levels, from which statistically significant training data and insightful information can be obtained. For client level backup job’s information only, more than 10 million time series data are available where each time series consists of historical data up to 3 years and contains rich information to be exploited for efficient enterprise backup management design.

In order to analyze the massive backup job data, which is also constantly expanding with new backup job information arrivals from operational data centers, we build a scalable data mining platform to store, process, and perform advanced data mining techniques on the overall data set, as illustrated in Figure 1. We leverage open source tools to reduce the overall cost while preserving flexibility. For backup job metadata storage, we leverage Cassandra NoSQL database due to its salient features of scalability and no central point of failure, among others [3]. The backup job metadata are replicated from the database server in a background process for data analytics purpose. The proof-of-concept Cassandra cluster consists of 7 nodes with SUSE Linux Enterprise Server and each node has 4 CPU cores, 8GB memory, and 100GB direct-attached disks, which emulates a small cluster using low cost commodity hardware. Note that our nodes are virtual machines and the allocated resources can be adjusted to experiment the impact of heterogeneous cluster hardware configurations on the performance of parallel data mining algorithms, which is another research focus of ours and beyond the scope of this paper.

To facilitate parallel data mining for the massive data set, we utilize Hadoop as the backend batch processing parallel computation platform, which is configured on the same set of nodes with Cassandra to facilitate data locality. In addition, we leverage the RHadoop framework [4] on top of our Hadoop and Cassandra cluster. RHadoop utilizes Hadoop streaming capability where each mapper and reducer function can be written in native R language instead of Java. Therefore, the rich capability of R in terms of statistic computing can be leveraged. In our scenario, we will store tens of millions time series data in Cassandra using customized schema, and the data mining tasks (for all time series data) are distributed to all work nodes of the Hadoop cluster in parallel. The platform can scale easily by simply adding commodity nodes with Cassandra, Hadoop client, and RHadoop packages installed. Our parallel data mining platform integrates with the backup analytics frontend tightly and provides graphical visualization to provide additional insights to the backup administrator via web-based management consoles.

III. MINING OF MASSIVE ENTERPRISE BACKUP DATA - USE CASE STUDIES

In this section, we discuss several massive data mining tasks that we are implementing on the big data analytics platform to facilitate advanced enterprise backup management capabilities.

A. Backup Capacity Planning Using Predictive Analytics

In the enterprise backup architecture shown in Figure 1, our backup analytics component contains rich historical information across multiple servers which can be leveraged for backup capacity planning using predictive analysis. At the storage backend, a designated primary storage pool is allocated for each backup server, which serves as a buffer to store incoming user data. To prevent buffer overflow, background tasks such as data compression, deduplication, and policy-driven information lifecycle management (ILM) processes are executed to reduce the utilized storage space. From a modeling perspective, the primary storage pool can be viewed as a G/D/I queue [5], as shown in Figure 2, where the arrival rate is the user data injection rate (from consecutive backup jobs), and the departure rate is determined by the ILM policy (e.g., migration frequency), the average compression ratio, the average deduplication ratio, among others. While the queue’s departure rate can be easily estimated and often configurable, the arrival rate of the primary pool essentially captures the
rate of backup data growth which is not controllable by the backup administrator. Therefore, in order to perform capacity planning of the primary storage pool, an accurate forecasting of the future data injection rate is desired. From the backup administrator’s perspective, over-provisioning the primary storage pool unnecessarily increases the backup cost, whereas an under-provisioning can potentially cause the primary storage pool being overwhelmed quickly to cause backup job failures. Therefore, our objective of the backup capacity planning is to provide predictive estimation of the backup data growth to strike the balance between risk and cost, which also facilitates a simpler budgeting procedure.

Fig. 2. A queueing model of the primary backup storage pool.

In order to predict the backup user data growth rate, we leverage the historical time series data of Backup Data Transfer Size, which is one of the over 20 attributes that we collected for each individual backup job, as the training set for our predictive model. In a similar context, [6] applies piecewise linear regression model to capture the capacity growing speed of backup data. In our work, we extend the predictive analysis by utilizing bagging (bootstrap aggregating) method, which is an ensemble method and has more robust and accurate predictions by aggregating multiple weak predictive models [7] trained by bootstrapping samples. Our implemented predictive models for bagging include (1) Weighted Linear Regression with LASSO: LASSO regulation is added to the weighted linear regression model to compensate overfitting, and the optimal penalty weight is determined using cross-validation; (2) Weighted Generalized Linear Regression (Poisson) with LASSO: Similar to above with Poisson link function; (3) Support Vector Regression: Similar to the standard SVM method applied for regression instead of classification; (4) Neural Networks: A three layer forward neural network; and (5) Random Forest [8]. All algorithms have been implemented in the RHadoop framework due to the volume of time series data and their suitability for parallel implementation. The predictive model training processes for all client level, server level, zone level time series of accumulative backup data sizes are distributed to all Hadoop nodes under the MapReduce framework. In addition, we periodically re-initiate the training process to update the prediction models, e.g., once a week. It is worth noting that our ensemble prediction framework enjoys the flexibility that each prediction algorithm is simply a pluggable building block and new predictive algorithms can be easily incorporated. Furthermore, the framework can optionally select a subset of algorithms (among all algorithms in the catalog) for quick training. For example, we select two clients and utilize their historical accumulative data backup size information during the past two months\(^1\) to train our ensemble model. Our prediction ensemble includes Linear Regression with LASSO, Support Vector Regression, and Neural Networks. Denote \(x(n)\) as the accumulative backup data size for a client at day \(n\) (assuming every client has daily backups). Our prediction objective is to learn the unveiled relationship between \([x(n-2), x(n-1), x(n)]\) and \(x(n+5)\). In other words, our goal is to predict the capacity occupation 5 days in advance given the observed capacity occupation values of the past three days.

Figure 3 shows the prediction of backup data size growth for two backup clients. The solid line shows the real size of total backup data and the lines with circles are the predictions by our ensemble method. Note that each prediction is made based on the data size observations of 5 days ago and our framework can be extended to long term growth rate prediction (i.e., predict the total backup data size beyond 5 days) provided that sufficient historical data are collected and trained. It can be observed from Figure 3 that our ensemble prediction maintains a balance between over-prediction and under-prediction and captures the backup size data growth. For more in-depth introduction of ensemble prediction principles, refer to [9].

B. Backup Anomaly Identification Using Contextual Outlier Detections

Outlier detection is also known as anomaly detection, novelty detection, deviation detection, fraud detection, or exception mining, in different contexts according to the applications. In general, an outlier is an observation (usually a data point with

\(^1\)In this paper, all absolute data values in our examples have been transformed to preserve anonymity.
possibly high dimension) that appears to deviate remarkably from other samples in which it occurs. While the concept is intuitive, a formal mathematical definition largely depends on the specific use case and application scenario for meaningful interpretations. Various outlier detection techniques have been studied in the literature, ranging from identifying outliers in streaming data, time series data, to graph anomaly detection in emerging social network applications. According to the underlying assumptions, existing outlier detection solutions can be divided into model-based and model-free methods. Popular model-based outlier detection algorithms include statistic methods where usually a parametric model is assumed and fitted using data, where an observation is identified as outlier if it is highly unlikely to occur according to the statistical model. In contrast, model-free methods such as density-based and clustering-based solutions have no assumptions on the model and can be utilized in an unsupervised fashion. For overview of state-of-the-art outlier detection techniques, refer to survey papers such as [10].

In the enterprise backup management context, several factors may cause a backup job to exhibit abnormal behaviors. For example, if a network device becomes defective, the backup jobs that traverse this device will experience much longer time to complete and the Job Duration metric will have larger values compared to previous backup jobs. Therefore, our backup analytics platform needs to capture such anomaly and alert the backup administrator for further problem determination and troubleshooting to prevent more severe consequences such as backup failures. The detection of outliers must be performed automatically, promptly, and accurately, by comparing the metadata of the newly recorded backup job with historical backup job information. In our framework, we emphasize on four key performance indicating metrics (among more than 20 metrics) to determine the degree of abnormality of each backup job, i.e., (1) data transferred (in KB), (2) average network throughput (in MBps), (3) the number of backup files, and (4) total job duration (in seconds) of the job. Upon the completion of each backup job, the metadata will be stored at the database server and replicated in our data analytics cluster for processing. Our outlier detection program, implemented in R and the Hadoop framework, will be invoked to first compute the outlier scores of each individual metric, say \( s_1, s_2, s_3, s_4 \), and a final outlier score, which measures the degree of abnormality of this backup job, is a weighted sum of \( \bar{s} = \sum_{i=1}^{4} w_is_i \) where the weight \( w_i \) has a default value of \( \frac{1}{4} \) and can be customized by the backup administrator to accommodate business-specific requirements. Note that other methods of combining outlier scores can be incorporated in our framework straightforwardly. To facilitate prompt outlier detection, we distribute the outlier score computation task (e.g., calculating the distances to neighbor points) across the underlying Hadoop cluster using the MapReduce framework where each node uses R packages for efficient data processing and calculation. Such a parallel implementation will significantly speed up the detection process in order to achieve a meaningful response time for the backup administrator to take appropriate countermeasures.

However, traditional outlier detection algorithms cannot be applied directly in our backup scenarios. For example, in the metric of “number of backup files,” the value varies significantly according to its backup job type. For “full” backup jobs, the average number of backup files is much larger than that of “incremental” backup jobs of the same backup client. In our scenarios, most backup clients have a mixture of backup job types specified by the backup policy, e.g., full backups at weekend and incremental backups during weekdays. In addition, the outcome of each backup job has a significant impact on the values of the aforementioned metrics. For example, a backup job may have an outcome marked as “partial” which suggests that the job finishes with an incomplete status, possibly due to an expiration of the backup window. Therefore, it is not uncommon that the number of backup files will be smaller compared to a backup job with a “successful” outcome from the same client. By leveraging the historical backup job information of each client, it is meaningful to identify the outliers in backup jobs with same type and outcome status. In other words, our objective is to identify contextual outliers where whether a single observation is an outlier or not depends not only on its value, but also on its context, i.e., its backup job type and its outcome. To achieve our goal, we extend a well-established point outlier detection algorithm Local Outlier Factor (LOF) [11], which is known to perform well in real world scenarios such as network intrusion detections [12] via extensive validations. In the following, we will use the metric of “number of backup files” as an example to illustrate how our contextual backup information can be incorporated. Denote \( f_n \) as the number of backup files for the new backup job and \( f = \{f_1, \cdots, f_{n-1}\} \) as the sequence of “number of backup files” for all backup jobs associated with the same backup client. Note that previous backup jobs for the backup client may have different outcomes and backup types. For \( f_n \), we first compute its \( k \)-distance with respect to \( f \) where \( k \) is a controllable parameter as will be discussed shortly. Formally, the \( k \)-distance of point \( f_n \) is the distance \( d(f_n, f') \) (Euclidean distance in our scenario) to another point \( f' \in f \), where \( f \subseteq f \) contains all backup jobs (from the same backup client) with the same outcome and backup job type with \( f_n \), and

1. at least \( k \) points of \( f'' \in f \) exist such that \( d(f_n, f'') \leq d(f_n, f') \), and
2. at most \( k-1 \) points of \( f'' \in f \) exist such that \( d(f_n, f'') < d(f_n, f') \).

Conceptually, the \( k \)-distance of \( f_n \) indicates the density around \( f_n \) with respect to the elements in \( f \) where a larger value suggests that \( f_n \) is further away from the rest of points and is likely to be an outlier. For notation brevity, we use \( N_k(f_n) \) to represent the set of points such that \( f' \in f \) and \( d(f_n, f') \leq k \)-distance of \( f_n \). For each data point \( f' \in N_k(f_n) \), define the reachability distance between \( f_n \) and \( f' \) as

\[
\text{reach-d}_{k}(f_n, f') \triangleq \max \{k\text{-distance}(f'), d(f_n, f')\}. \quad (1)
\]
The local density of \( f_n \) is defined as
\[
lrd_k(f_n) = \frac{|N_k(f_n)|}{\sum_{f' \in N_k(f_n)} \text{reach-d}_k(f_n, f')}
\]
which is the reciprocal of the average reachability distance of \( f_n \) to its neighbors in \( N_k(f_n) \). Therefore, points that are in the “center” of a cluster should have larger values of local density whereas points further away from other neighbors will have smaller values of local density. Finally, define the local outlier factor (LOF) outlier score of point \( f_n \) as
\[
LOF_k(f_n) = \frac{1}{|N_k(f_n)|} \sum_{f' \in N_k(f_n)} \frac{lrd_k(f')}{lrd_k(f_n)}
\]
where the value of \( f_n \) will be incorporated to compute \( lrd_k(f') \). Therefore, for each new backup job’s metric, i.e., the number of backup files in our example, our extended algorithm will assign an outlier score to it, and from (2), it is apparently that most “normal” points will have an outlier score around 1 whereas an abnormal point will have a larger value of its outlier score. The further an observation deviates from its neighboring points, the larger outlier score it has. Therefore, by simply selecting an outlier score threshold, we are able to identify whether the new backup job’s number of backup files exhibits abnormal behaviors. The degree of abnormality of this backup job can be obtained by a weighted sum of the outlier scores of key performance indicating metrics such as the number of backup files, the job duration, the amount of data transferred, the average network throughput experienced, among others. The algorithm is unsupervised and data-driven which requires no assumption on the data statistical distribution. Also, the framework is suitable for our Hadoop/MapReduce implementation since the computation of reachability distances to all neighbors in \( N_k(f_n) \) and their local density in (2) can be performed in a parallel fashion due to their uncoupled nature. The only controllable parameter of the algorithm is \( k \), which is the number of neighbors \( f_n \) should be compared to for outlier score computation. Unfortunately, there is no systematic way to obtain the optimal value of \( k \) and it has been shown in [11] that the computed outlier score is not monotonic with respect to \( k \). Therefore, we set a range of \( k \) from \( k_{\text{min}} = 10 \) to \( k_{\text{max}} = 20 \) and compute the outlier score of \( f_n \) for each \( k \). The maximum outlier score is used as the final outlier score associated with \( f_n \).

As an example, Figures 4 shows the top 6 outliers that are identified by our extended algorithm within the most recent 60 backup jobs in terms of (1) number of backup files, (2) data transferred, (3) network throughput, and (4) job duration, for a particular backup client. The top 6 outliers (sorted in a decreasing order of the outlier score) are labeled in circles whereas the rest of points are labeled in stars. Note that the examples are for demonstration purpose only to show the effectiveness of the contextual outlier detection solution. From an operational perspective, when a new backup job information is collected and recorded in the database, our outlier detection algorithm will compute the outlier score of the job, which is a weighted sum of the outliers scores of the four aforementioned performance metrics, with respect to the historical backup job information, e.g., in the last two months. If the overall score exceeds a predetermined threshold, e.g., combined outlier score > 2, our backup analytics system will generate an alert to the backup administrator. The outlier score computation leverages the parallelism and our Hadoop platform integrated with R packages. It is worth noting that both the outlier score weights and the anomaly alerting threshold can be adjusted by the backup administrator. Our on-going research work is to enhance the outlier detection mechanism by adjusting the threshold automatically according to the backup administrator’s actions to the reported outliers.

C. Backup Failure Prevention Using Frequency Mining

A backup job can fail for various reasons, ranging from a simple network device failure to more complex scenarios where the root cause needs to be revealed by scrutinizing the detailed logs. From a business continuity’s perspective, a backup job failure increases the risk and the adverse impact of data loss, and hence a preventive analysis of backup failures is extremely valuable for the backup administrator. In practice, certain types of backup failures are preceded by specific patterns of events, a.k.a., signatures. For example, in the network failure example, the dysfunctional network device may yield long processing delay and low throughput before dropping the traffic completely. Therefore, prior to the backup job which fails due to the unreachable network device, earlier backup jobs that traverse the same network device will experience consecutive long delay and low network bandwidth, which can be observed in the backup analytics platform by mining backup job information across multiple servers. If such patterns are identified before the device goes defunct completely, the backup administrator can take preventive actions such as reconfiguring the backup servers to use an alternative network path. Note that identifying such patterns may require a thorough trace analysis across multiple backup servers and millions of time series data needs to be analyzed. In our work, we extract a failure training set from all historical data where each element in the set is the sequence of backup job logs (e.g., up to 10 days) prior to three consecutive backup job failures for the same client. We assume that three consecutive backup failures for the same client is a critical event and needs to be prevented. Within the failure training set, we implement a customized frequency mining solution to identify the most frequent patterns and events in preceding sequences of each three-day-consecutive-failures occurrence. We model all the backup jobs in a single day as an event, where the metadata such as the network throughput and the number of backup files are the items within each event. Our objective is to identify the most frequent appeared patterns, i.e., sequences of items, within 10 days window before a three-day-consecutive-failures occurs.

There are several frequency mining algorithms that have been proposed in the literature, which can be divided into two categories depending on whether the order of each pattern is preserved. One type of frequency mining algorithms such as FP-Growth and Eclat [13], [14] identify the group of
frequent items as an unordered collection. For example, in a retailer scenario, customer purchase history can be analyzed and products that are frequently purchased together will be identified for informative business decisions. The other type of frequency mining algorithms aim to identify the group of frequent items with a particular order. Our backup failure prevention problem falls into the second category. We utilize SPADE algorithm [15] which is essentially a breath-first search algorithm and has been widely adopted for sequential frequency mining applications. The algorithm starts from the most frequent singleton pattern and then evolves by adding more events satisfying a frequency threshold. In our algorithm, we set the frequency threshold to 0.8, which suggests that only the patterns that appear more than 80% from all samples with three consecutive failures will be pruned and presented. Next, our solution utilizes customized MapReduce implementations to distribute the computationally intensive tasks to multiple commodity compute nodes for parallelism and scalability. For example, the appearance counting process of each sequence is distributed to multiple work nodes in a similar fashion as the “WordCount” example [16] where each mapper function is the pruning process using SPADE algorithm and the reducer function is simply the calculation of overall appearance frequency within the dataset. Note that frequency mining has been used for backup block retrieval in [17], which differs from our work in terms of application and their centralized computation scheme. Our solution framework implements the frequency mining tasks for more than half of million backup clients (with a frequency threshold of 0.8) utilizing our RHadoop framework introduced in Section II. Due to space constraints, we omit the results of anonymized frequently appeared patterns in our managed backup environments.

IV. SUMMARY AND DISCUSSIONS

In this paper, we introduce our on-going research effort of leverage big data analytics in the enterprise IT management domain. In specific, we investigate the backup management solution design from a scalable data mining perspective. Our proof-of-concept platform integrating big data techniques with enterprise backup analytics architecture is introduced. We also discuss several use case studies as examples of leveraging big data mining solutions for enterprise backup management and industrial IT management in general.

In this paper, we extend the LOF algorithm for outlier detection due to its simplicity and wide applicability. There are several improvements on the LOF algorithm such as [18]–[20] to further reduce its complexity and worth our further investigation. For example, in [21], an online version of LOF algorithm is proposed which can further reduce the computation complexity. Another our on-going research focus is to integrate our backup outlier detection framework with active learning capabilities [22], i.e., incorporating domain knowledge of backup administrators into our decision framework. We allow the backup administrator to label the identified outliers with “uninteresting outliers” and “interesting outliers” tags as in [23]. Therefore, an additional binary classifier is introduced which can actively learn from the backup administrator by selecting the most uncertain outliers for human classification. The iterative process will refine the selection of outliers by incorporating field experience and domain expert knowledge to capture intangible aspects.

REFERENCES


