RC25159 (W1105-031) May 13, 2011
Computer Science

IBM Research Report

CARDIO: Cost-Aware Replication for Data-Intensive WorkflOws

Claris Castillo, Asser N. Tantawi, Diana Arroyo, Malgorzata Steinder
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

CARDIO: Cost-Aware Replication for Data-Intensive workflOws

Claris Castillo, Asser N. Tantawi, Diana Arroyo and Malgorzata Steinder
IBM T. J. Watson Research Center
19 Skyline Drive, Hawthorne, New York 10532
Email: {claris, tantawi, darroyo, steinder}@us.ibm.com

***Abstract*—In this work we are concerned with the cost associated with replicating intermediate data for dataflows in Cloud environments. This cost is attributed to the extra resources required to create and maintain the additional replicas for a given data set. Existing data-analytic platforms such as Hadoop provide for fault-tolerance guarantee by relying on aggressive replication of intermediate data. We argue that the decision to replicate along with the number of replicas should be a function of the resource usage and utility of the data in order to minimize the cost of reliability. Furthermore, the utility of the data is determined by the structure of the dataflow and the reliability of the system. We propose a replication technique, which takes into account resource usage, system reliability and the characteristic of the dataflow to decide what data to replicate and when to replicate. The replication decision is obtained by solving a constrained integer programming problem given information about the dataflow up to a decision point. In addition, we built a working prototype, CARDIO of our technique which shows through experimental evaluation using a real testbed that finds an optimal solution.**

***Index Terms*—replication, dataflows, map-reduce, Hadoop, data-availability, fault-tolerance**

I. Introduction

The landscape of data intensive processing has evolved significantly in the last few years. Such processing has now become much more pervasive and is accessible to a broader user population. Several factors are responsible for this development. First, there is tremendous growth in the volume of available data resulting from the proliferation of devices [1]. Second, the data storage costs have reduced dramatically making it cost-effective for institutions and individuals to retain large volumes of data. Third, new programming paradigms, such as Map-Reduce [2] and Pig [3], have recently emerged that enable efficient processing of large data sets on clusters of commodity hardware. Open-source implementations of these paradigms such as Hadoop [4] have further promoted the democratization of data-analytics.

Commodity computing is a key enabler in the the development and success of large-scale data analytics in a Cloud environment. This paradigm enables "scaling out" by adding inexpensive computing nodes as a solution to the scalability problem. This has resulted in frequent failures that have became a rule rather than an exception in typical Cloud environments. For example, in the context of data analytics, Google has reported 5 average worker deaths per Map-Reduce job [2], and at least one disk failure in every run of a 6-hour Map-Reduce job on a cluster of 4,000 machines [5]. Not surprisingly, fault tolerance is considered a primary goal in the design and development of middleware and application software that processes data on such a large scale. The performance degradation resulting from failures as well as the cost for handling such failures depends on the nature of the application and its corresponding requirements.

In this work we are concerned with failures that result in unavailability of intermediate data in the execution of dataflows. The availability of intermediate data is crucial to the performance of dataflows since lost intermediate has to be regenerated for the dataflow to advance. Therefore, in order to recover from a single failure, multiple stages that were previously executed in the dataflow may have to be re-executed again. This *cascaded re-execution* effect has shown to be responsible for significant degradation in the end-to-end performance of Map-Reduce jobs in production systems [6]. This problem exacerbates for dataflows with large number of stages [7] since it increases the probability of failure per job and the corresponding failure recovery time.

Replication is one mechanism that has been widely used to improve data availability in data-intensive applications. Together with scheduling and placement strategies this technique has proven successful in Grid environments [8]–[11]. Cloud environments, however, follow a consumption-based business model [12]. Hence, replication techniques that have proven successful for Grid environments in the past may not be cost-effective for data-analytics in Cloud environments. As a matter of fact, paid Cloud replication services already exist for traditional applications [13], [14]. We expect the emergence of similar services for data analytic applications.

We argue that the cost paid to provide for reliability in large-scale data analytic environments may be excessive if this cost is not well understood. This is crucial as system providers seek to reduce cost by relying on commodity hardware to build their infrastructures. The key question we seek to answer in this work is: given a dataflow with a set of stages when is cost-effective to replicate its corresponding intermediate data?

To address this question, we introduce a metric to capture reliability cost. This metric represents the price paid by either the infrastructure or the user to handle failures that results in unavailable input data of a given stage. This cost is effectively comprised of two costs: *Regeneration cost*, which is the price paid for re-executing a task for which output data has been lost and re-execution is needed to advance in the dataflow.*replication cost*, which represents the price paid for creating replicas of the data in order to reduce the likeliness of losing data in the presence of failures. In the former, the price is a measure of the amount of resources needed to re-execute stages. In the latter, the price is a measure of the resources needed to store and maintain replicas in the system. In practice, it is challenging to understand the cost of regeneration for a given dataflow in the presence of failure due to the data and temporal dependencies between stages. Therefore, to provide for data availability, existing data analytic platforms implement replication techniques that may not be appropriate to users. Furthermore, such techniques usually assume over-provisioning of storage resources and tend to over-replicated. For instance, in a typical Hadoop cluster intermediate data of a dataflow is replicated thrice by default. We expect the cost of replication to increase drastically with the high projected growth in data volumes [15] and with the growing trend towards shared data analytic environments [16]–[18].

In this work we propose CARDIO a technique and a system which minimize the reliability cost. We formulate the minimum reliability cost problem as an as an integer programming optimization problem with nonlinear convex objective functions. The optimal solution to this problem dictates the replication factor of intermediate data upon completion of its corresponding stage without *a priori* knowledge of future (downstream) stages. To find such a solution, CARDIO takes into account the probability of loosing data, the cost of replication, the storage capacity available for replication and potentially the current resource utilization in the system. More importantly, upon completion of a stage in the dataflow, CARDIO reconsiders the replication decisions made at the previous stages to ensure that it achieves a local optima at each stage while satisfying the storage constraint.

We implemented CARDIO as a decision layer on top of Hadoop that makes intelligent replication decisions as the dataflow advances towards completion. In summary, this work makes the following major contributions:

- We introduce a formulation of the minimum reliability cost problem as an optimization problem for dataflows.
- Provide key insights into the cost of reliability of dataflows from a resource perspective.
- A novel cost-aware replication system that incarnates an efficient solution to the minimum reliability cost problem.

Structure of the Paper

We first start by providing background information useful to understand our work in Section II. We then present a quantitative analysis of the cost of replication for data-intensive workflows in a real Hadoop-based environment in Section III. In Section IV we formulate the minimum-cost reliability problem for dataflows. In Section V we perform a numerical analysis of CARDIO. In Section VI we describe the architecture of the CARDIO system, a system that encapsulates our solution and is based on a real Hadoop Cloud testbed. In addition we perform an evaluation of our system on real workloads. We then discuss the related work in Section VII. In Section VIII we conclude the paper and present our future plans in Section IX.

II. Background

We chose Hadoop to drive our experimental work since it is the de-facto data-analytic platform in use today by the industry and the academic community. Within this framework we consider a data-analytic platform in where dataflows consist of compute stages where each stage is a Map-Reduce job (tuple). Note, that our work applies to any data-analytic platform which relies on replication to provide for data-availability of dataflows. To help understand our work better we provide some basic background on Hadoop.

A. Hadoop Map-Reduce

The Hadoop Map-Reduce engine is a Java-based platform developed by Yahoo! that supports the Map-Reduce programming model [2]. A Map-Reduce job specifies a map function, which transforms a job input record into a set of key/value pairs, and a reduce function, which transforms an intermediate key and the set of all values associated with that key into the job output record.

The Map-Reduce layer uses a cluster of nodes (machines) to run Map-Reduce jobs. In Figure 9 we show the Hadoop architecture with some additional components related to our system. One special node runs the `JobTracker`, which organizes the cluster's activities. The other nodes each run a `TaskTracker`, which organizes a worker node's activities. The `TaskTracker` is configured with a number of map and reduce tasks to run concurrently, and pulls new tasks from the `JobTracker` as old ones finish. Each task is run in a separate process. A job is organized into two sequential tiers of *map tasks* and *reduce tasks*. The Hadoop Map-Reduce layer stores intermediate data produced by the map tasks and consume reduce tasks in the local filesystems of the machines running the map and reduce functions.

B. Hadoop Distributed File System (HDFS)

HDFS is a distributed filesystem developed by Yahoo!. It is designed to provide high streaming throughput to large, write-once-read-many-times files. A HDFS filesystem requires one unique server, the *Primary-Namenode* and a *Secondary-Namenode* as backup. The filesystem is built from a cluster of *DataNodes*, each of which serves up blocks of data over the network using a HTTP based protocol. These components are shown in Figure 9. Note that DataNodes are denoted by DN.

HDFS is designed to run on a collection of commodity machines, thus is designed with fault tolerance in mind. A file is stored as a sequence of blocks; typical block sizes are 64 MB and 128MB. Each block is replicated across multiple Datanodes. The default replication value is three (3). Replicas are distributed so that one copy is local to the client and two other copies are stored remotely.

Replication is performed in a pipelined fashion. Data is first written to a local file. When the local file accumulates a full block of user data, the client retrieves a list of DataNodes from the NameNode. This list contains the DataNodes that will host a replica of that block. The client then flushes the data block to the first DataNode. The first DataNode starts receiving the data in small portions (4 KB), writes each portion to its local

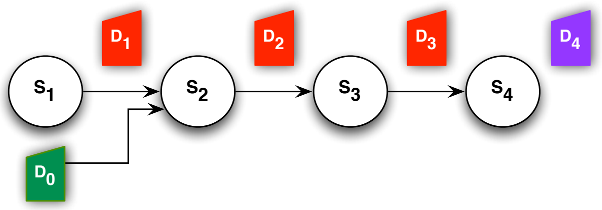


Fig. 1: A dataflow consisting of four stages: Tagger, Join, Grep and RecordCounter.

repository and transfers that portion to the second DataNode in the list. This process repeats until the last replica is created. Consequently, a file can not be accesses till all replicas are done.

C. Testbed

We use a real testbed for our experiments consisting of a Hadoop-0.20.2 cluster with 25 VMs hosted in an internal Cloud-environment available to IBM Research. Each VM has RHEL5.3, 4-2.9GHz CPUs and 350 GB of storage.

D. Workloads

To drive our experiments we use a real Map-Reduce dataflow consisting of 4 stages: *Tagger*, *Join*, *Grep* and *RecordCounter*. Figure 1 depicts these stages as S_1, S_2, S_3 and S_4, respectively.

- Tagger pre-processes by tagging an input data set consisting of records. This is a IO intensive workload that outputs 170GB of data as input to the second stage.
- Join consists of a standard table join operator that joins the table output by *Tagger* with a dimensional table previously stored in the HDFS. The fact and dimensional table are 170GB and 115GB, respectively. *Join* is I/O intensive but more CPU intensive as compared to Tagger.
- Grep is considered a representative workload for data analytics and consists of a simple operation that look for records that match a given regular expression in the output of the Join stage. This stage generates 100GB.
- RecordCounter counts all records with a common field. Its output is of 70GB.

III. Cost-Benefit Analysis of Replication in Dataflows

We ground our work by first quantifying the cost resulting from replication of intermediate data in dataflows. We consider end-to-end completion time for this study. We recognize that any measurement used in this evaluation depends on the particular replication technique and various characteristics of the system. Without any loss of generalization, the results can be safely extrapolated to include other platforms. Note that in this evaluation we do not seek at providing an study of the impact of failures in dataflows. For such study in the context of Map-Reduce tuples we refer the reader to the work [6]. Instead, we are interested in quantifying the price paid to provide for data availability in dataflows or in other words: the reliability cost.

The following discussion pertains to three types of performance costs. (1) Replication cost, which represents the cost of replicating intermediate data in a dataflow expressed in terms of resource usage, time or any other relevant metric; (2) regeneration cost, which estimates the cost to regenerate the intermediate data in case of failure; and (3) reliability cost, which considers both the replication cost and the regeneration cost in estimating the effective cost of handling failures in the system. Note that we assume that data that has been lost is regenerated upon it is needed as input to a stage. We formally define these costs later in Section IV.

RFM	t_1	t_2	t_3	t_4	T
NR	1823	2759	4547	3759	12888
1100	4022	5097	5219	4346	18684
0101	1824	5703	5493	5465	18485

TABLE I: Completion time in seconds for individual MR stages and end-to-end time of dataflow.

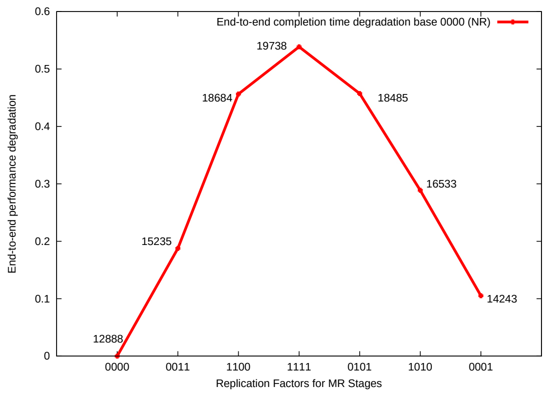
We define a replication factor map (RFM) as the sequence of replication factors for all the stages in a dataflow. For example, given a 4-stage dataflow, an RFM of 0011 corresponds to the case in which only the third and fourth stages of the dataflow (corresponding to D_3 and D_4 in Figure 1, respectively) were replicated. We consider 0000 and 1111 two special cases of RFM as they implement two extremes replication strategies: no-replication (NR) and full-replication (FR), respectively.

A. Replication Cost and Benefit

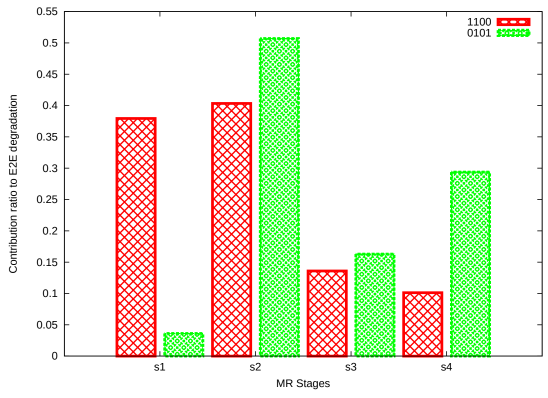
We analyze the end-to-end performance of our 4-stage dataflow described in Section II-D under various replication strategies. To do this we execute our dataflow with a pre-defined RFM and measure its end-to-end completion time. Since our purpose is to quantify the cost of replication for our dataflow we calculate the degradation resulting from replicating using RFM=0000 (NR) as a baseline. We plot our results of average of 3 rounds in Figure 2(a). In this Figure x-axis depicts the RFM of choice.

As observed from Figure 2(a), data availability comes at a cost to the end-to-end performance depending on the level of replication considered. This degradation peaks at 0.53 when the replication strategy is FR. This behaviour follows intuition since intermediate data is replicated for all intermediate stages for this case. For other RFM values such as 0001 and 0010, the performance degradation is much lower, since these cases result in the replication of a smaller amount of intermediate data as compared to when the replication strategy is FR. We can further observe that RFM values of 1100 and 0101 show approximately same degradation ratios. This result is counter-intuitive since for RFM=1100 the intermediate data replicated is about 1.58 times larger than for RFM=0101 (see Section II-D).

To investigate the causes behind this observation we breakdown the completion time of the dataflow. Table I plots the execution time for the four stages and the end-to-end job completion time for the RFM values of NR, 1100 and 0101. Figure 2(b) plots the fractional contribution of each stage to the total increase in end-to-end job completion time. When we compare the two cases of RFM=1100 and RFM=0101, we observe that–as expected– the completion time of S_1 increases for the case RFM=1100 when compared to RFM=0101. It is noticeable that although S_1 only contributes 14% to the



(a)



(b)

Fig. 2: (a) End-to-end performance degradation due to replication. (b) Data-locality achieved due to replication.

total time of the dataflow in the NR strategy (see Table I) its completion time constitutes about 39% of the total degradation of the dataflow when D_1 is replicated for RFM=1100 (see Table I). Different conclusions can be reached from these observations depending on which cost metric is considered. If the cost metric of interest is time –as it is in our case– the re-execution cost outweighs the replication cost since it takes longer to replicate D_1 than to re-execute S_1 later if a failure cause D_1 to be lost. If on the other hand, we consider a case wherein there is a limited budget for a given resource, e.g., CPU, associated with the dataflow, then it is plausible that regeneration cost outweighs the replication cost and hence replicating D_1 is the right decision.

B. *Summary*

The conclusions that can be inferred from our motivational exploration are multifold. First, the cost of providing reliability for dataflows varies depending on the chosen replication strategy. With de-facto full-replication available in standard distributed file systems such as HDFS, costs as high as 50% are possible when time us considered as metric of cost. Second, that when making replication decision one should carefully consider the trade off between the cost of replicating intermediate data and the cost of regenerating that data in the presence of failure. Third, that the choice of cost metric is crucial in making replication decisions in dataflows.

With these observations, we investigate the cost of reliability for dataflows. More specifically, we are interested in finding for a given stage what is the replication decision that leads to a minimum reliability cost for the dataflow with no-knowledge of downstream stages. In the next section we formulate the minimum reliability cost optimization problem to address this question.

IV. Problem Definition and Analysis

A. *Problem statement*

Consider job J (also referred to as a dataflow) which consists of n stages in sequence $S_i, i = 1, 2, \cdots, n$. Stage S_i uses data D_{i-1} generated from its predecessor stage S_{i-1} and generates data D_i to be consumed by its successor stage S_{i+1}. (S_0 and S_{n+1} are not real stages, rather they refer to the input and output processes of job J.) It is assumed that data D_0 is available as input to job J and D_n is its output. The storage size of data D_i is denoted by Y_i storage units and the time it takes to store D_i is $E_i = \delta Y_i$, where δ is the speed of storing a data unit. (We use time and cost interchangeably.) At the conclusion of stage S_i, data D_i is stored as a copy referred to as the original copy. Further, additional $x_i \geq 0$ copies of D_i are stored as replicas of the original copy in case the latter (or any of the replicas) is lost due to failure. Note that D_0 is assumed to be reliably stored. The processing time, i.e. the time needed to process D_{i-1}, perform any necessary computation during S_i, generate D_i, and store an original copy is denoted by A_i. Further, the replication time needed to store the x_i replicas is denoted by R_i, where

$$R_i = x_i E_i.$$

The storage devices where data (original or replica) is stored are subject to failures. We assume the following *failure model*. Consider a piece of data D_i that is stored as $b_i \geq 1$ blocks. The number of blocks b_i is a function of the block size o_i and the size of teh file Y_i. More specifically, $b_i = \frac{Y_i}{o_i}$. A single block fails independently of other blocks with probability p. A file is corrupted if at least one of its blocks fails. Given $x_i \geq 1$ as the number of replicas, all replicas are corrupted, hence D_i is unavailable (failed), if at least all x_i replicas of a given block fail. Let $f_i(x_i)$ denote the conditional failure probability of D_i. Then, we have

$$f_i(x_i) = 1 - (1 - p^{x_i})^{b_i}.$$

As long as at least one copy (original or replica) of D_{i-1} is available, Stage S_i proceeds in its processing unaffected. However, if all copies of D_{i-1} are lost, stage S_{i-1} is reinvoked in order to regenerate data D_{i-1} and its replicas. Denote by G_{i-1} the expected time to regenerate D_{i-1} and its replicas ($G_0 = 0$). Let the expected total stage time for S_i be T_i. It is given by

$$T_i = A_i + R_i + G_{i-1}.$$

The expected regeneration time G_i is given by

$$G_i = f_i(x_i+1)T_i$$

since $f_i(x_i+1)$ is the probability of losing all x_i+1 copies of D_i (replicas and original) and T_i is the expected time to regenerate D_i and store it along with x_i replicas. Note that T_i includes any potential loss of data of predecessor stages in a recursive manner.

The total job execution time, denoted by T, is the sum of all n stage times,

$$T = \sum_{i=1}^{n} T_i$$

which includes the total processing time, the total replication time, and the total expected regeneration time due to failures of all n stages, i.e.

$$T = A + R + G$$

where A is the job processing time, R is the job replication additional cost (penalty), and G is the job expected regeneration additional cost, each is given by

$$A = \sum_{i=1}^{n} A_i, \quad R = \sum_{i=1}^{n} R_i, \quad G = \sum_{i=1}^{n-1} G_i,$$

respectively. (Note that a variation of the above definition of the regeneration cost G may include G_n in case the job output D_n is also subject to failure which necessitates its regeneration. In such a case we have $G = \sum_{i=1}^{n} G_i$.) We define the *reliability cost*, Z, as the additional cost due to replication and regeneration,

$$Z = R + G. \tag{1}$$

The total storage needed for all replicas is given by

$$Y = \sum_{i=1}^{n} x_i Y_i.$$

Note that the choice of the replication vector $X = [x_1 x_2 \cdots x_n]$ impacts the values of the replication and regeneration additional costs, R and G, respectively, as well as the storage need, Y. Intuitively, the more replicas, the higher the replication cost and storage need, and the lower the regeneration cost. This gives rise to an optimization problem in order to determine an optimal value of X. However, before describing the optimization problem, we discuss the temporal aspect of data replication.

B. Dynamic Replication

So far, we considered the replication factor, x_i, for stage S_i as a static variable during the entire duration of the job. In general, one might change the value of x_i dynamically as the job progresses through its various stages. For instance, reducing the replication factor of an earlier stage as higher stages execute may make sense in order to allow more recent data to be replicated. To allow for dynamic replication, we extend the notation as follows. When the job finishes executing stage S_k, we say that the job is in step $k, k = 1, 2, \cdots, n$. The replication factors at step k are denoted by $x_i(k), i = 1, 2, \cdots, k$. In other words, after stage S_k completes, data generated at stage $S_i, i = 1, 2, \cdots, k$, is replicated with a factor $x_i(k)$, leading to a lower triangular matrix of variables, denoted by $\mathbf{X}$. Thus, dynamic replication gives rise to $n(n+1)/2$ replication factor variables.

An increase from $x_i(k)$ to $x_i(k+1)$ means that data D_i needs more replicas. Whereas, a decrease means giving up storage space taken by D_i, potentially in favor of replicating more "valuable" data. The replication cost at step k is given by

$$R(k) = x_k(k)\ E_k + \sum_{j=1}^{k-1} (x_j(k) - x_j(k-1))^+\ E_j,$$

where the first term is the replication cost of stage S_k and the second term is the additional replication cost from step $k-1$ to step k due to any increase in the replication factors of stages $S_j, j = 1, 2, \cdots, k-1$. We assume that removing (demoting) replicas does not incur any significant cost. (Note that the second term is zero for $k = 1$ and $x_j(0) = 0$.)

The expected regeneration cost at step k involves the handling of potential failures of data D_{k-1} during the execution of stage S_k, i.e.

$$G(k) = f_{k-1}(x_{k-1}(k-1)+1)\ \ T_{k-1}(k-1),$$

where

$$T_i(k) = A_i + R_i(k) + G_{i-1}(k),$$
$$R_i(k) = (x_i(k) - x_i(k-1))^+\ E_i,$$

and

$$G_{i-1}(k) = f_{i-1}(x_{i-1}(k-1)+1)\ \ T_{i-1}(k-1).$$

Let $Z(k) = R(k) + G(k)$. The total replication and regeneration costs are given by

$$R = \sum_{k=1}^{n} R(k), \quad G = \sum_{k=2}^{n} G(k),$$

respectively, and their sum is Z as given in 1. At step k, the storage constraint is given by

$$Y(k) = \sum_{i=1}^{k} x_i(k)\ Y_i \leq C,$$

where C is the total available storage capacity for replication.

C. Optimization Problem

The assumption of whether the parameters $(n, A, Y, \cdots)$ of job J are know *a priori* or only the parameters related to each stage become known at (or about) the completion of each stage is crucial in determining the nature of the optimization problem. Accordingly, we differentiate between two optimization criteria: *job optimality* and *stage optimality*.

1) Job optimality (JO): In *job optimality* we assume the knowledge of all job parameters before the job starts. In such a case, the objective would be to choose the replication matrix $\mathbf{X}$ so as to minimize the total expected cost Z subject to replication storage constraint. Thus, the problem is

$$\min_{\mathbf{X}} Z \quad s.t. \quad Y(k) \leq C, \quad k = 1, 2, \cdots, n.$$

2) Stage optimality (SO): In *stage optimality* we assume that the parameters related to each stage become known at (or about) the completion of each stage. In such a case, at step k, which coincides with the completion of stage $S_k, k = 1, 2, \cdots, n$, the decision variables $X(k) = \{x_i(k), i = 1, 2, \cdots, k\}$ are determined. Note that, at step k, one may alter earlier choices of replication given the current conditions and available storage. The criterion is to minimize the incremental replication cost and the regeneration cost in case a failure occurs to D_k. Thus, the problem at step k is

$$\min_{X(k)} Z(k) \quad s.t. \quad Y(k) \leq C.$$

Given the equations above, we note that both the job and stage optimality problems are integer programming problems with nonlinear convex objective functions. In the subsequent sections we obtain the optimal solution numerically.

D. Minimizing resource utilization

The above problem formulation is stated with time (processing time, replication time, regeneration time, etc.) as the measure of concern. More generally, each stage of the job involves usage of various resources. For instance, the execution of a stage consumes CPU (user as well as system cycles), disk (read and write operations), network (transmit and receive packets), and storage of produced data and its replicas. The usage of each of these resources may incur costs that differ from stage to stage during the job execution. Hence, we may be concerned with minimizing cost rather than time. The problem formulation remains the same, though.

Let K be the number of resources used during the execution of a stage. (In the above example we have $K = 4$, since we considered CPU, disk, network, and storage resources.) Denote by $u_{i,k}, k = 1, 2, \cdots, K$, the usage of resource k during stage S_i in units of that resource. In order to make usage uniform among the resources, we normalize usage by defining $\rho_{i,k}$ as

$$\rho_{i,k} = \frac{u_{i,k}}{\sum_{j=1}^{n} u_{j,k}}, \quad k = 1, 2, \cdots, K,$$

so that $\sum_{i=1}^{n} \rho_{i,k} = 1$ for all resources. The relative costs of resources is represented by weights $\omega_k, k = 1, 2, \cdots, K$, in such a way that one resource (say $k = 1$, without loss of generality) has a standardized unit weight. Thus, the unit cost is the total job usage of that resource. The weights of the other resources are expressed in terms of their relative costs to the cost of the standardized resource. Thus, A_i in the above problem formulation is given by

$$A_i = \sum_{k=1}^{K} \omega_k \, \rho_{i,k}.$$

Similarly, R_i may be expressed as the cost of replication. Hence, we would be minimizing total job cost, replication cost, and regeneration cost, instead of total job time, replication time, and regeneration time. In either case, the variables are the same, namely the number of replicas taken at the conclusion of each stage of the job.

To summarize, in this section we have provided a formulation to the minimum reliability cost for dataflows that motivate this work. In particular, we are interested in the SO formulation since it assumes no *a priori* knowledge of the dataflow and hence, it is more suited to realistic settings.

V. Analysis

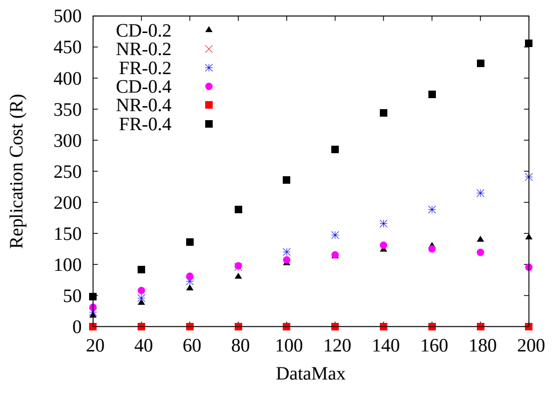
In this section we perform a parametric analysis to evaluate the optimal solution to the SO problem as it compares to standard replication strategies adopted in practice such as full-replication FR. We consider various parameters of interest in real systems and investigate their trade-offs.

Performance metric We use *replication cost* (R), *regeneration cost* (G) and *reliability cost* (Z), as defined in Section IV, as the comparative cost metrics. Recall that R represents the cost of generating the initial replicas and does not include the cost involved in creating new replicas when intermediate data is re-generated in the presence of failure. G involves the cost of re-execution of stages due to failures and their corresponding replication decisions. Z is the sum of R and G and corresponds to the overall cost of providing reliability in the system. In many of our experiments we only plot G and R since Z alone does not suffice to differentiate the strategies in terms of performance. While two strategies may have similar values for Z, they might differ significantly in their value of R and G. In other words, the cost of reliability may come at different replication and regeneration cost.

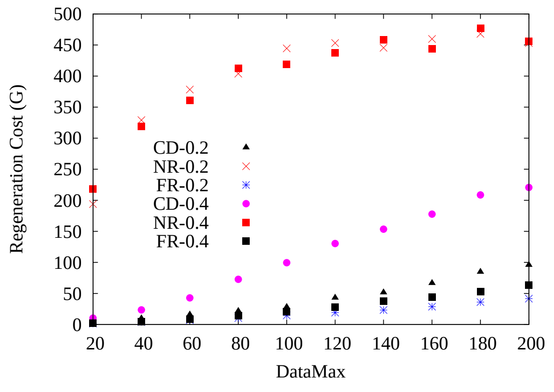
Recall that the input to the problem is a dataflow consisting of multiple stages in sequence. For the ease of analysis and with no loss of generalization, we use 4 stages represented as S_1, S_2 , S_3 and S_4. Note that experiments with larger number of stages show similar results. Similar to Section IV, we use time as the cost metric for our evaluation. A stage S_i is represented by the tuple $< D_i, C_i >$ where D_i and C_i corresponds to output data size and computing resources of the stage, respectively. In our analysis we assume that one unit of computing resources requires one unit of time. The system is represented by the tuple $< \delta, p >$ where δ corresponds to the time required by the system to replicate one data unit and p is the probability of a replica being unavailable due to failure as described in Section IV.

We compare CARDIO with two baseline strategies of full-replication(FR) and no-replication(NR). Recall that FR corresponds to the default replication technique used in existing Hadoop-based platforms, in which intermediate data is replicated thrice for every stage. NR corresponds to the strategy in a Hadoop cluster where replication is disabled.

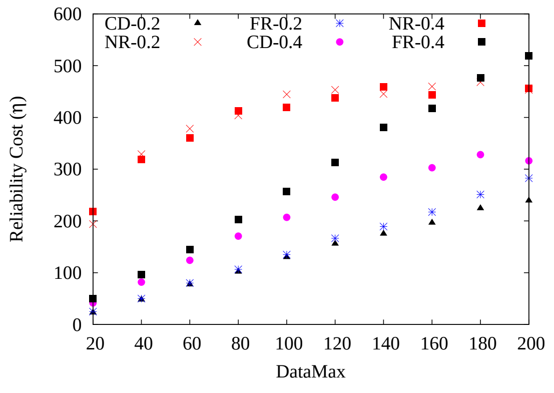
In our analysis–unless stated differently–we consider a typical scenario with the following configuration: $p = 0.2$, $C = \infty$ and $\delta = 0.2$. The value of δ was chosen so that R does not outweigh G. $p = 0.2$ characterize a reasonable reliable system. Later in this section we investigate the impact of each individual parameter on the performance of the replication strategies. Our reasoning for choosing C is that by relaxing the storage constraint for replication we aim at reproducing typical Hadoop clusters that are over-provisioned in terms of storage. For each stage S_i, C_i is obtained from an uniform distribution $U(1, C_{Max})$ where $C_{Max} = 100$ unless specified otherwise. Similarly, D_i is also obtained from a uniform distribution $U(1, D_{Max})$ where D_{Max} may vary within the



(a)



(b)



(c)

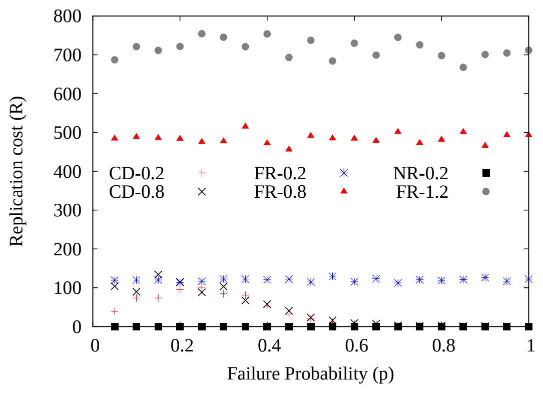
Fig. 3: Impact of D_{Max} on the performance of replication strategies.

range $[1, C_{Max}]$ in order to control the ratio between storage and computing resources required for a dataflow.

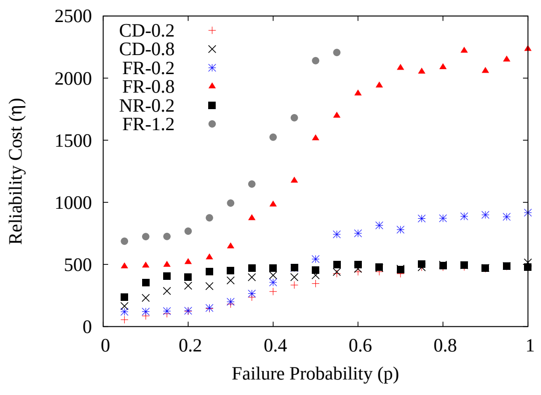
Effect of D_{Max} We first investigate the impact of varying the ratio between storage and computing resources on the performance of the replication strategies. Figure 3 plots the varying costs for FR, NR and CARDIO as we vary D_{Max} for various values of δ. For example, "CD-0.2" in the key refers to the CARDIO strategy with $\delta = 0.2$. Multiple observations can be drawn from these results. First, Figure 3(a) shows that as the amount of intermediate data increases, the reliability cost (Z) increases steadily for both FR and CARDIO . This result is expected since a larger amount of intermediate data yields higher replication costs, which in turns increases G and R. For NR, on the other hand, there is no replication cost involved, and therefore $R = 0$ while G and Z increase as the probability of failure of the dataset increases due to the larger value of D_{Max}. Second, for small values of δ (0.2), both CARDIO and FR perform similarly as shown by the complete overlap of their corresponding Z curves. As observed in Figures 3(b) and 3(c), CARDIO achieves smaller R as compared to FR, while FR outperforms CARDIO in terms of regeneration cost G. Such behaviour is a side-effect of the higher failure probability resulting from larger values of D_{Max}. That is, as the failure probability increases, CARDIO sees diminishing returns of replication since replicas are likely to be lost due to failures. As a result, CARDIO replicates more lazily under such conditions. This, in turn, results in higher values of G as observed in Figure 3(c).

Effect of failure probability Figure 4 evaluates the impact of p on CARDIO's performance as compared to FR and NR, by presenting G and Z for the three strategies and various values of δ. It can be observed that R increases as δ increases. This fact is a consequence of the higher cost paid for replication. Furthermore, for FR the curve for Z lays above the corresponding curves for NR and CARDIO for all values of p. The adoption of FR has the potential of being very costly under conditions where the cost of replicating data is high. Figure 4 (a) shows that when replicating data is inexpensive, e.g., $\delta = 0.2$, however, FR and CARDIO performs similarly for $p < 0.5$ (observe the full overlap of both curves) and CARDIO outperforms FR for $p \geq 0.5$. This fact is supported by the same argument presented for the previous experiment. That is, as p increases CARDIO sees diminishing results from replicating and replicates more conservatively. In a nutshell, CARDIO effectively emulates FR (NR, respectively) under conditions where replication is inexpensive and (expensive, respectively) and various levels of reliability of the storage system. To gain a better insight of how these parameters relate in Figure 5 we plot a 3-dimensional plot with p and δ in the x-axis and y-axis, respectively. δ is presented in the z-axis. The plot shows that for low values of p and δ CARDIO offers its best performance since R and G both remains low. When p is high and δ is low, Z is dominated by a high regeneration cost (G). Thus, the cost of computation A determines the behaviour and performance of CARDIO. It follows intuition that there is a configuration for δ and p for which both G and R have equal values. This point determines the flipping point at which CARDIO switch modes and either replicate aggressively (FR) or avoid replication (NR) and it can be observed $\delta = 1$ and $p = 0.25$ in Figure 5. CARDIO is able to varies its performance between the performance achieved by NR and FR by intelligently adapting to the conditions of the system while minimizing the reliability cost (Z).

Effect of storage constraints We evaluate the performance



(a)



(b)

Fig. 4: Impact of failure probability (p) on the performance of replication strategies.

of CARDIO under various storage constraints, i.e., for varying values of C. For this purpose, we introduce a new parameter σ such that $C = \sigma \times \sum_i^N Y_i \times x_i$, where $N = 4$ corresponding to the 4 stages in our example and $x_i = 3$ representing FR. σ represents the fraction of the maximum capacity needed for CARDIO to fully replicate, i.e, emulate FR. Figure 6 plots R and G as a function of σ for various values of δ. Note that for the sake of clarity we only show the results for CARDIO. From Figure 6 (a), it is observed that R increases with σ, i.e., increases with larger C value. This follows intuition since CARDIO can replicate more data with increase in storage. Nevertheless, as the cost of replication increases ($\delta > 0.4$) R flatten out. This result shows that CARDIO uses the storage capacity available for replication efficiently while trying to minimize the overall reliability cost. It is also noticeable that for $\delta = 0.8$ CARDIO is insensitive to σ. This is expected since CARDIO avoids replicating if R is too high. In Figure 6 (b) we observe that this fact results in a expensive G.

Effect of block size Figure 7 plots the performance of CARDIO and FR as a function of block size o for various values of δ (0.1, 0.3 and 0.5). For a given D_i as o_i increases, the number of blocks (b_i) –across which the intermediate data set is stored– decreases and therefore the probability of loosing an intermediate data set ($f(x_i)$) due to failure also decreases. Refer to Section IV for a description of the failure model. As explained earlier, CARDIO replicates more lazily under such conditions since it does not achieve much gain from replicating data in a fairly reliable system. This observation is supported by the steady decrease in value of R observed in the plot presenting R (not included due to space constraints). On the other hand, FR is oblivious to varying block size and/or probability of failure. Due to space constraints we only plot Z in Figure 7 for this evaluation. We observe that Z remains relatively constant for FR in Figure 7 while CARDIO outperforms FR consistently across the spectrum of b except when δ is very small ($\delta = 0.3$). These results are consistent with our observations in Figure 5.

Finally, recall that CARDIO demotes upstream replicas in order to accommodate for downstream replicas and to maintain a minimum value of Z as the dataflow advances. This metric does not represent a deficiency of Cardio. Instead it depicts mechanisms by which Cardio is able to adapt to the storage constraint while optimizing for Z. Figure 8 plots the average number of demoted replicas as we relax the storage constraint, i.e., by increasing σ. We observe that the number of demoted replicas increases as σ increases and peaks when $\sigma < 0.4$ for all values of δ. This behavior can be explained as follows. For smaller values of σ, CARDIO has limited storage available for replication and hence it is forced to demote a larger number of replicas in order to minimize Z as a dataflow advances. Additionally, note that as δ increases the average number of demoted replicas start peaking at smaller values of σ. This is a consequence of the lower replication factor of CARDIO when the replication cost (δ) is high.

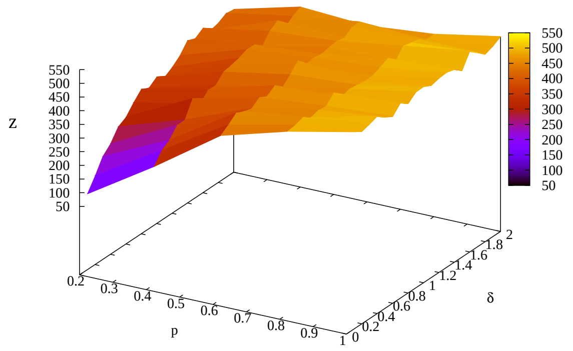
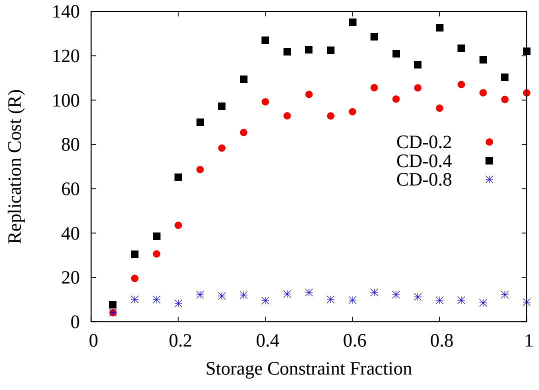


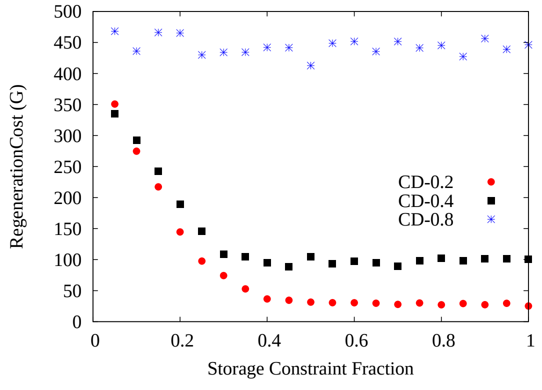
Fig. 5: Impact of δ and p on performance of replication strategies.

VI. CARDIO System

We have implemented a full working system named CARDIO which incarnates the technique presented in the previous section. More specifically, CARDIO is a decision making framework that makes intelligent decisions regarding the replication of intermediate data for dataflows with the objective of minimizing the cost of reliability. In this section we describe the CARDIO architecture and implementation. Our prototype is implemented over the experimental testbed described in Section II-C. Nevertheless, the underlying principles behind it apply equally to any dataflow engine.



(a)



(b)

Fig. 6: Impact of storage constraint (σ) on performance of replication strategies.

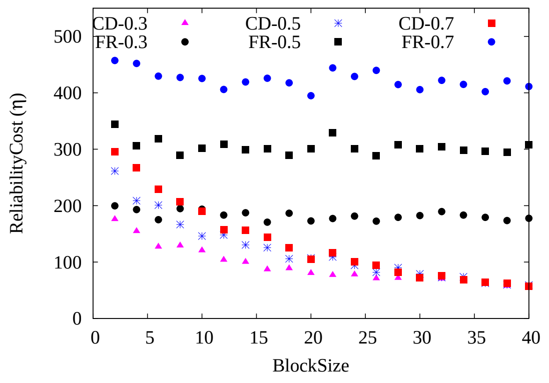


Fig. 7: Impact of block size (o) on the performance of replication strategies.

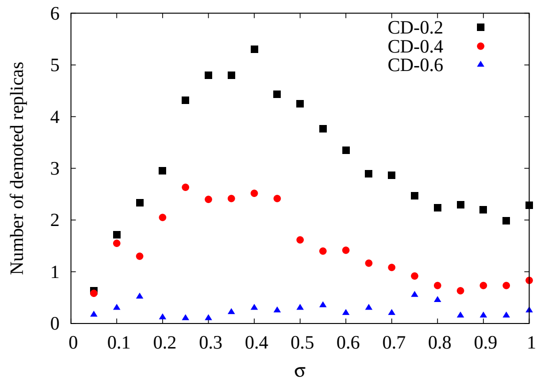


Fig. 8: Average number of replicas demoted in CARDIO.

Figure 9 shows a high-level design of our CARDIO framework. CARDIO is a feedback control-loop system that is composed of three major components: a set of *sensors* (CardioSense), a *controller* (CardioSolve) and an *actuator* (CardioAct). Following we describe each component in detail.

CardioSense One of the driving motivations behind CARDIO is to enable resource aware replication, i.e., cost metric captures resource consumption due to reliability. CardioSense component is responsible for collecting resource usage statistics for running stages in the cluster and for the cluster itself. To collect such statistics, CardioSense relies on monitoring processes (*HMon*) hosted in each individual worker node. HMon continuously monitors Map-Reduce tasks. These statistics are accumulated and served to CardioSense upon request. HMon is a Python-coded tool based on *Atop* [19] with negligible overhead ($< 2\%$ CPU utilization).

CardioSense periodically contacts the *JobTracker* to obtain the current progress of stages that are active or running in the cluster (step 1 in Figure 9). When a stage has made enough progress and reaches a pre-configured threshold value, CardioSense contacts all the HMon processes in the cluster (step 2 in Figure 9) and aggregates the received accounting information for the given job. This is done via a SOAP client.

CardioSolve CardioSolve is the heart of the CARDIO system. It implements the solver for the SO problem introduced in Section IV. CardioSolve receives as input resource usage data from from CardioSense. Upon completion of a stage, CARDIO uses this data to arrive at an optimal solution consisting of a replication factor vector to include the recently completed stage as well as all previous stages upstream (step 3 in Figure 9. In other words, CARDIO reconsiders decisions made in the past for previous stages (step 4 in Figure 9) effectively demoting or promoting replicas when needed.

CardioAct Once CardioSolve arrives to an optimal solution, the replication factors of all the completed stages have to be updated to reflect the RFM in the solution. To facilitate these modifications CardioAct must implement a client of the storage layer to handle modification requests. In our prototype, CardioAct implements the HDFS Java client and uses the API call `setReplication(Path file, short newReplicationFactor)` at the time of its invocation (step 5 in Figure 9) to modify the replication factor for each data set as needed(step 6 in Figure 9).

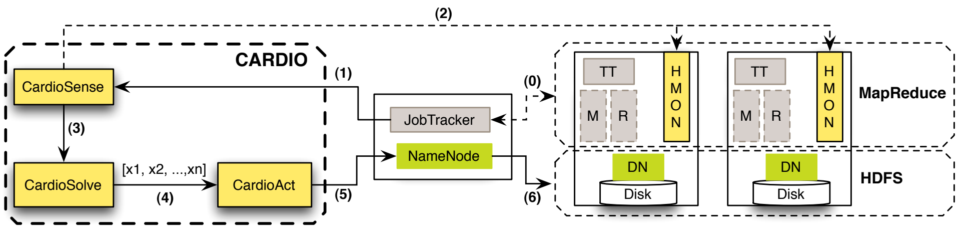


Fig. 9: Architecture of the CARDIO System.

Resource/Stage	S_1	S_2	S_3	S_4
CPU_S (cycles)	98764200	1661784	327588400	2272171
CPU_U (cycles)	16801600	197759	73120700	709814
NET_R (bytes)	499482639	148368227457	482358793261	290260079138
NET_W (bytes)	129352834157	147517875661	55905501611	50007380711
DSK_R (bytes)	104200	168192	7870811200	102824128
DSK_W(bytes)	12744	16264	1018375200	123496648
STG (bytes)	170G	170G	100G	70G

TABLE II: Resource usage for dataflow. CPU_S and CPU_U stand for system and user CPU utilization, respectively. NET_R and NET_W represent the number of bytes received and sent over the network. DSK_R and DSK_W correspond to the number of bytes read from and written to disk, respectively. STG refers to the storage requirement of intermediate data output for a given stage and is known once the primary copy of the intermediate data has been stored.

A. Experimentation

In this section we show the results of preliminary evaluation of the CARDIO system. Due to space constraints we focus only on two main aspects in our evaluation. First, we want to examine with the applicability and usefulness of CARDIO in the context of Cloud analytic providers. Therefore, we evaluate CARDIO's ability to utilize resources efficiently under resource-constrained conditions. Second, we are interested in examining the performance benefit perceived by the user when replication is done conservatively only if needed. This benefit depends on the replication technique used since the overhead incurred by replication varies according to the technology. We quantify this overhead in Section III.

We drive our experiments using the Map-Reduce dataflow described in Section II-D. To compute A_i for every stage we use the resource usage information obtained from HMon under the NR strategy. Table II presents the resource usage information for each stage of the dataflow. We develop a simple common resource usage metric to use in CARDIO to solve the SO problem. We normalize the usage for each resource by dividing each column in Table II by the sum of its corresponding row. This process is explained in Section IV-D.

In our experiments we want to evaluate CARDIO's ability to consider resource usage information available from the system while making replication decisions. Intuitively, re-executing stages that stress an already over-loaded resource can potentially hinder the performance of the system. This negative effect can be effectively prevented in CARDIO by attaching weights to the actual computing cost of stages equivalent to factors that reflect the utilization of resources in the system. If we consider an example scenario of CPU constrained conditions, the CPU cost associated with each stage of a dataflow can be scaled up by a factor to reflect the higher importance of this resource. In response to this, CARDIO will tend to replicate intermediate data corresponding to such stages to minimize the regeneration cost (G).

We do not create resource constrained conditions in the system. Instead we add hooks to CardioSense that allow it to report various resource constrained conditions. When CardioSense reports a resource bottleneck, CardioSolve uses weights to increase the importance of that particular resource for the job (Section IV-D). We use CPU++, NET++, DSK++ and STG++ to represent a scenario where CPU, network, I/O and storage resource is over-utilized respectively and hence should be treated as an expensive or scarce resource. For our evaluation, we consider the following configuration: $p = 0.08$, $\delta = 0.6$ and $C = 0.4$ (204GB). Note that δ represents the replication cost and it depends on various system characteristics that include replication technology and storage. The value of δ is fixed in our prototype. Nevertheless, for a production system we expect this value to be obtained from profiling and historical information from the the system.

Figure 10 plots the results of this evaluation. Note that x- and y-axis represent specific scenarios and the end-to-end completion time, respectively. Table III also shows the optimal solutions provided by CARDIO upon completion of each stage in the dataflow. As observed from Figure 10, the end-to-end completion time of the dataflow under NR strategy is the smallest. However, the end-to-end completion time for FR strategy ramps up by a factor of 1.62. These results are consistent with the results presented in Section III.

CPU++ scenario. When CardioSolve reports CPU as a constrained resource, CARDIO only replicates the third stage (Table III). This follows intuition since S_3 is the stage with the highest CPU consumption (by an order of magnitude more). **NET++/DSK++ scenario.** When the network is constrained after completion of S_2 CARDIO decides to replicate to later revert its replication decision (demote replica) to accommodate for S_3 and S_4. Recall that DSK follows a usage similar to NET because of the presence of intermediate data between the map and the reduce phase. Thus, the solution of CARDIO for NET++ and DSK++ are similar.

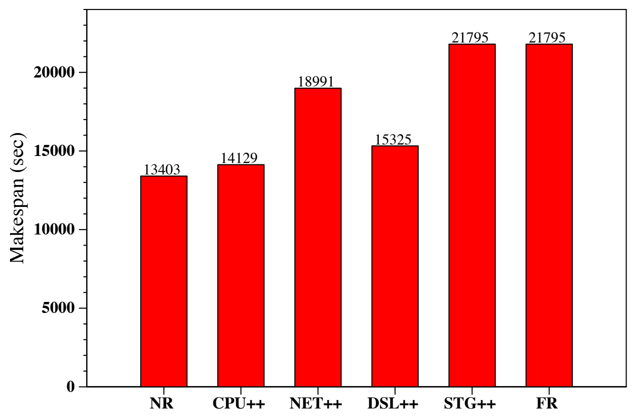


Fig. 10: CARDIO vs NR/FR under various workloads.

STG++ **scenario.** We consider the case where storage is expensive in this scenario. In this case, CARDIO behaves similar to FR, i.e., it replicates after completion of every stage. However, due to the storage constraint imposed in the configuration (C=240GB) CARDIO demotes every replica (Table III). The reasoning behind this observation is that when storage is expensive, stages with large intermediate data sets have high regeneration cost. However, replication of their corresponding intermediate data sets will quickly exhaust the storage allocated for replication (C). As a result, such replicas are likely to be demoted as the dataflow progresses downstream while CARDIO seeks to satisfy the storage constraint.

We showed in Section V that adopting CARDIO in a highly unreliable environment is equivalent to adopting the NR strategy, i.e, very limited benefit is attained from replication. Our current virtual cluster consists of relatively new hardware making it impossible to evaluate the performance of CARDIO in the presence of failure under realistic conditions. Therefore, we reproduce the probability model introduced earlier in Section IV. We modified HDFS so that it successfully retrieves each block with some probability p upon reading an input data set. p is an input parameter to the system. As a reminder, when an HDFS client fails to retrieve a block, it attempts to find a replica for the given block in a remote rack. If no replica is available, the file read operation fails and the HDFS client is informed by means of an exception. A compute stage that receives such an exception when reading its corresponding input data set triggers the execution of the upstream stage in order to re-generate the input intermediate data. Figure 11 plots the end-to-end completion time for our experiment for traditional Hadoop with 3 replicas per file (denoted by Hadoop-FR3) and for CARDIO when CPU is the resource bottleneck. Each data point corresponds to an average of three experimental runs. Our results show that CARDIO is able to reduce the end-to-end completion time by up to 70% for $p = 0.1$. A closer look at our data traces shows that various stages were re-executed due to the injected failures through out the execution of the dataflow. Stage S_2 re-executed more frequently due to its large input data set. Also, our traces show that S_1 for $p = 0.06$, $p = 0.08$ and $p = 0.1$ was re-executed 3, 1 and 21 times, respectively. Note that CARDIO achieves its performance gain from replicating only S_3 with $x_3 = 1$ as

CPU++	NET++	DSK++	STG++
0	0	0	1
00	01	00	01
001	001	001	001
0010	0011	0011	0011

TABLE III: Decision vectors for CARDIO under various resource usage configurations.

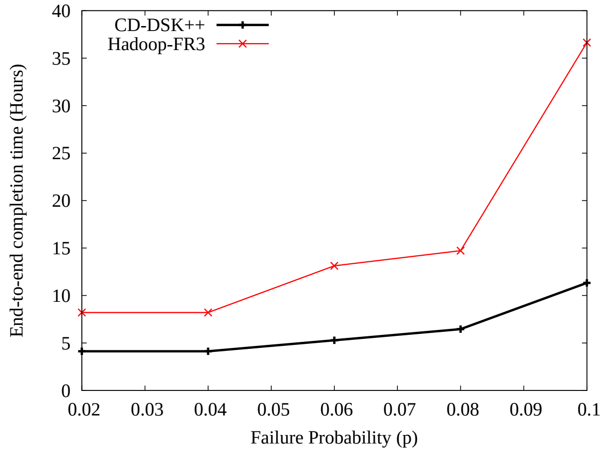


Fig. 11: CARDIO vs Hadoop FR strategy for a real workload.

shown in Table III. The results were found to be similar under any other resource contention scenario.

VII. Related Work

Data availability for intermediate data of dataflows has received much attention recently due to great interest around data analytics . In [6] the authors look at the problem of data availability for intermediate data of workflows. In particular, an Intermediate Storage System is proposed that seeks at minimizing the impact of server failures on the availability of intermediate data. In this work this is achieved by proposing improved replication techniques that minimize the interference resulting from replication. This is in contrast to our work which aims at making better replication decision. In fact, both contributions complement each other in that one aims at replicating only when it is entirely necessary and the other one improves the efficiency of the replication mechanism.

Replication as a mechanism for reducing access latency and bandwidth consumption for files in Grid environments has been studied in the past [20]. This work focuses on replicating files based on their access patterns and placement. Much research has been done on the replication of files for improving access time. For instance, in [21] a simulation framework is used to investigate replication strategies based on a cost model that evaluates data access costs and performance gains of creating each replica. In a similar direction, in [22] a collection of strategies is proposed to reduce access time. Similar to CARDIO, in this work the authors propose an algorithm to decide on when to replicate a file. This decision, however, is based on whether the replication will result in a reduced expected access time in the future. One work that resembles in goal to CARDIO is [23] in that it investigates various techniques to maintain data availability of files in a peer-to-peer system above a threshold by pro actively creating replicas in a decentralized fashion. Note that all aforementioned works in Grid consider the problem of file replication. This is in contrast to CARDIO which addresses the problem of minimizing reliability cost for dataflows where there are strict dependencies in between datasets. This difference clearly distinguishes CARDIO from the aforementioned works.

The problem of cost replication has been widely studied in the context of replica placement in WAN settings for Web proxies and caches [24]–[26]. CARDIO addresses the problem of when to replicate.

VIII. Conclusions

In this work we have considered the problem of minimizing the cost of reliability for dataflows by balancing the cost of replication and regeneration of stages in the presence of failure. We formulated this problem as an integer programming optimization problem with non-linear convex objective function. We used standard solving techniques to solve this problem and analyze its performance against standard replication techniques used in state-of-the art data analytic platforms.

We realized our approach into a real system called CARDIO. CARDIO is a novel replication system for dataflows that incarnates the aforementioned ideas and has been prototyped in a real data-analytic testbed with real workloads. CARDIO departs from traditional replication frameworks in that it weighs the regeneration cost against the replication cost in order to decide if intermediate data needs to be replicated. Our results show that CARDIO is effective in finding an optimal solution to the minimum reliability cost problem.

IX. Discussion

CARDIO is the materialization of our initial exploration on the problem of minimizing the cost of reliability for dataflows. Our solution is unique in that it introduces the minimum reliability cost optimization problem for dataflows. CARDIO is our first step towards building a robust decision making system for efficient and effective replication for dataflows.

To simplify the understanding of our analysis we consider dataflows with a maximum fan-in degree of 1. This configuration is not uncommon in existing MapReduce production systems [7]. As a matter of fact, a common structure for data-analytic programs is to subject data to a series of filtering operations [3]. We plan on extending our current solution to tackle more general cases.

One additional benefit of replication in data analytic platforms is data locality. We touched upon this scenario using a running example in Section III. Optimizing the placement of replicas is crucial to fully leverage this benefit. There is a clear trade-off between the cost of replication and data-locality in terms of the end-to-end performance of dataflows. We plan on investigating extensions to the CARDIO architecture to accommodate effective data locality.

In our current setup, we assume that the cost involved in demoting and maintaining replicas is negligible. We believe that these costs will become important in due to the Cloud business model. In the same path, a recurrent question to us is which entity is responsible for paying the price of handling reliability in such environments. The service provider or the client. This quest opens the door to interesting resource management problems in the context of this work. We will explore some of these issues as part of our future work.

CARDIO is reactive in that replication is driven by the completion of data stages. We plan to investigate more proactive approaches to replication. Finally, we would like to investigate the performance of CARDIO under various failure models.

References

[1] IDC, "The Diverse and Exploding Digital Universe," IDC, Tech. Rep., 2008.

[2] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *OSDI*, San Francisco, CA, US, December 2004, pp. 1–10.

[3] C. Olston, R. Benjamin, U. Srivastava, R. Kumar, and A. Tomkins, "Pig Latin: A Not-So-Foreign Language for Data Processing," in *SIGMOD*, Auckland, New Zeland, June 2008.

[4] Apache, "Apache Hadoop," http://hadoop.apache.org/.

[5] J. MacCormick, N. Murphy, M. Najork, C. A. Thekkath, and L. Zhou, "Boxwood: Abstractions as the Foundation for Storage Infrastructure," in *OSDI*, San Francisco, CA, 2004, pp. 8–8.

[6] S. Y. Ko, I. Hoque, B. Cho, and I. Gupta, "On Availability of Intermediate Data in Cloud Computations," in *HotOS*, San Diego, CA, June 2009, pp. 1–10.

[7] "Yahoo! presentation," http://wiki.apache.org/hadoop/HadoopPresentations.

[8] K. Ranganathan and I. Foster, "Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications," in *HPDC*. IEEE, 2002, pp. 352–259.

[9] S. Pandey and R. Buyya, "Scheduling of Scientific Workflows on Data Grids," in *CCGRID*, IEEE, Ed., Lyon, France, May 2008, pp. 548–553.

[10] S. Bharathi and A. Chervenak, "Data Staging Strategies and Their Impact on the Execution of Scientific Workflows," in *DADC*. New York, NY, USA: ACM, 2009, p. 5.

[11] H. Lin, J. Abawajy, and R. Buyya, "Economy-Based Data Replication," in *eScienceIEEE*, 2006.

[12] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud Computing and Grid Computing 360-Degree Compared," in *GCEGrid Computing Environments Workshop*, Austin, Texas, January 2008, pp. 1–10.

[13] https://wiki.duraspace.org/display/duracloud/DuraCloud.

[14] http://www.informaticacloud.com/solutions/data-replication.html.

[15] A. Thusoo, S. Anthony, N. Jain, R. Murthy, Z. Shao, D. Borthakur, J. S. Sarma, and H. Liu, "Data Warehousing and Analytics Infrastructure at Facebook," in *SIGMOD*, Indianapolis, IN, 2010.

[16] "Infochimps," http://infochimps.org.

[17] IBM, "ManyEyes," http://manyeyes.alphaworks.ibm.com/manyeyes/.

[18] "NASA Nebula," http://nebula.nasa.gov/.

[19] "Atop performance monitor," http://freshmeat.net/projects/atop/.

[20] K. Ranganathan and I. Foster, "Identifying Dynamic Replication Strategies for a High-Performance Data Grid," in *Proceedings of the International Grid Computing Workshop*, 2001, pp. 75–86.

[21] H. Lamehamedi, B. Szymanski, Z. Shentu, and E. Deelman, "Data Replication Strategies in Grid Environments," in *ICA3PP*, 2002.

[22] W. H. Bell, D. G. Cameron, L. Capozza, A. P. Millar, K. Stockinger, and F. Zini, "Simulation of Dynamic Grid Replication Strategies in OptorSim," in *Journal of High Performance Computing Applications*. Springer-Verlag, 2002, pp. 46–57.

[23] K. Ranganathan, A. Iamnitchi, and I. Foster, "Improving Data Availability through Dynamic Model-Driven Replication in Large Peer-to-Peer Communities," in *CCGRID*. Washington, DC, USA: IEEE Computer Society, 2002, p. 376.

[24] H. Yu and A. Vahdat, "Minimal Replication Cost for Availability," in *PODC*, Monterey, CA, 2002.

[25] L. Qiu, V. N. Padmanabhan, and G. M. Voelker, "On the Placement of Web Server Replicas," in *Proceedings of IEEE INFOCOM*, 2001, pp. 1587–1596.

[26] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman, "Placement Algorithms for Hierarchical Cooperative Caching," in *SODA*, Baltimore, MD, 1999.