2012 32nd IEEE International Conference on Distributed Computing Systems

ADAPT: Availability-aware MapReduce Data Placement for Non-Dedicated Distributed Computing

Hui Jin, Xi Yang, Xian-He Sun, Ioan Raicu
Department of Computer Science
Illinois Institute of Technology
Chicago, IL, USA
{hjin6, xyang34}@iit.edu, {sun, iraicu}@cs.iit.edu

Abstract— **The MapReduce programming paradigm is gaining more and more popularity recently due to its merits of ease of programming, data distribution and fault tolerance. The low barrier of adoption of MapReduce makes it a promising framework for non-dedicated distributed computing environments. However, the variability of hosts resources and availability could substantially degrade the performance of MapReduce applications. The replication-based fault tolerance mechanism helps to alleviate some problems at the cost of inefficient storage space utilization. Intelligent solutions that guarantee the performance of MapReduce applications with low data replication degree are needed to promote the idea of running MapReduce applications in non-dedicated environment at lower costs. In this research, we propose an Availability-aware DAta PlacemenT (ADAPT) strategy to improve the application performance without extra storage cost. The basic idea of ADAPT is to dispatch data based on the availability of each node, reduce network traffic, improve data locality, and optimize the application performance. We implement the prototype of ADAPT within the Hadoop framework, an open-source implementation of MapReduce. The performance of ADAPT is evaluated in an emulated non-dedicated distributed environment. The experimental results show that ADAPT can improve the performance by more than 30%. ADAPT achieves high reliability without the need for additional data replication. ADAPT has also been evaluated for large-scale computing environment through simulations, with promising results.**

Keywords-MapReduce, Reliability, Performance

I. INTRODUCTION

The ever-growing demand in data processing and computing power of technical computing nowadays always exceeds the capabilities of individual institutions. The advances of cyberspace have enabled distributed resource sharing to fulfill increasing application demands [1] [2] [3]. Distributed resource sharing environments take advantages of donated/voluntary resources in the Internet to harness computing power. Currently, only limited compute-intensive applications can benefit from non-dedicated distributed computing. The loosely-coupled nature and the lack of trust among the nodes in distributed systems make it extremely difficult to use standard parallel programming models such as MPI [4]. The inherent complexity and poor fault tolerance of standard parallel programming models further impede their applicability for distributed computing environments.

MapReduce is an emerging programming model that simplifies parallel data processing [5]. Due to the feature of easy programming, the fault tolerance support, and the assumption of commodity hardware, MapReduce has been recognized as a compelling programming paradigm for non-dedicated distributed computing environment to expand the scope of supported applications [6].

The primary concern faced by distributed resource sharing environments is the non-dedicated feature and the associated volatility. Each participating host can be turned off and leave the system at the owner's will. The problem is further exaggerated by the fact that non-dedicated distributed systems usually preempt external applications with local/host jobs. For example, SETI@home performs external scientific applications in the form of screen savers [1]. Another system, Condor, terminates the external tasks whenever keyboard or mouse events are detected [3].

MapReduce cannot eliminate the inherent volatility of non-dedicated distributed computing. However, several features of MapReduce make it unique from traditional scenarios:

- Hosts comprise of both CPU and storage resources
- Non-dedicated distributed computing systems generally have poor network connectivity

MapReduce applications are usually data-intensive rather than compute-intensive. Each participating host contributes not only CPU cycles, but also storage space to external applications. Storage space is a more sensitive resource than CPU since it constantly consumes disk space until the completion of external applications. A large storage space of the external applications could impact the disk usage and impact the fidelity of resource owners.

Non-dedicated distributed computing systems usually utilize broadband network connectivity, with significantly lower performance than dedicated systems. The uplink speed of a typical Internet host is usually less than 1Mb/s, and the download speeds less than 15Mb/s (compare this with bi-directional 1Gb/s network connectivity or better for dedicated systems). As a consequence, the migration of a single MapReduce data block (e.g. 64MB) could take up to several minutes, which significantly degrade the application performance.

The MapReduce model was conceived with the principle that "moving computation is much cheaper than moving data". Locality is a primary criteria for task scheduling. As such, the data placement decision guides the MapReduce task scheduling and eventually affects the application performance. The optimization of data placement in the area of non-dedicated computing is new and not yet well exploited by existing studies [7] [8].

Existing MapReduce framework randomly distributes data blocks onto each node. This mechanism works well for typical cluster environment that assumes homogenous hosts. However, nodes in non-dedicated distributed computing environment are usually heterogeneous in both computing power and availability, which makes the existing random data placement strategy inappropriate.

While the heterogeneity in computing power has limited impact on data-intensive MapReduce applications, hosts in non-dedicated computing environment may present considerable heterogeneous features in availability and hurt application performance. Table 1 summarizes the Mean-Time-between-Interruptions (MTBI) and interruption durations based on the data collected from SETI@home [9]. We can observe that the standard deviation of each parameter is usually several folds higher than its mean, which leads to a high value of Coefficient-of-Variance (CoV) and confirms the availability heterogeneity issue in non-dedicated distributed computing environments.

Table 1: Summary of SETI@home Data

	Mean	Std Dev	CoV
MTBI (seconds)	160290	701419	4.376
Interruption Duration (seconds)	109380	807983	7.3869

In this paper, we propose ADAPT, an *Availability-aware DAta PlacemenT* strategy for MapReduce to mitigate the impact of volatility and heterogeneity. ADAPT dynamically dispatches data blocks onto participating hosts based on their availabilities. ADAPT helps to mitigate the impact of vulnerability without increasing the replication degree, improve the data locality of MapReduce applications, and inherently reduce the network traffic.

The contribution of this research is three-fold.

1. We propose an analytical model to estimate the execution time of MapReduce tasks under non-dedicated distributed computing environments. Based on the model, an availability-aware data placement algorithm is presented.
2. We implement the prototype of ADAPT by extending Hadoop, an open-source implementation of MapReduce [10]. ADAPT incurs minor overheads to the existing Hadoop framework. It is designed to be an add-on feature of Hadoop and can be enabled/disabled flexibly.
3. Extensive experiments and simulations have been conducted to evaluate the feasibility and payoffs of ADAPT. We have built an emulated non-dedicated distributed computing environment on Magellan [11], the cloud computing platform at Argonne National Laboratory. The simulation was performed based on the failure traces collected from production systems [9]. The experimental results show that ADAPT improves application performance by more than 30%. ADAPT also achieves comparable a performance while delivering better storage efficiency.

The rest of this paper is organized as follows. Section II introduces the background of this study. In Section III, we present the analytical model that estimates the MapReduce task execution time under non-dedicated distributed computing environment. Based on the model, an availability-aware data placement algorithm is introduced to build the basis of ADAPT. Section IV introduces the design and implementation of ADAPT on Hadoop. The performance evaluation is presented in Section V. Section VI reviews the related work. We conclude this study with future work in Section VII.

II. Background

A. MapReduce

MapReduce is a framework for processing large scale dataset, which mainly consists of two steps: Map and Reduce [5]. Users specify the Map and Reduce functions and the underlying runtime system automatically parallelizes the computation. Map tasks take *key-value* pairs as input and convert them into intermediate *key-value* pairs, which are used by reduce tasks to generate output.

The MapReduce framework relies on scalable distributed file systems such as GFS [12] to store the input, intermediate results and the output.

MapReduce was initially developed by Google for web search on large-scale commodity hardware. Recent studies have revealed the potential of MapReduce to support applications in other areas such as machine learning [13], bioinformatics [14], high energy physics [15], and cosmology [16].

In addition to Google's MapReduce implementation, several other systems have also been developed to implement the MapReduce framework. Leading examples include open-source Hadoop [10] and Microsoft Dryad [17].

B. Hadoop

Hadoop is a Java based open-source implementation of MapReduce [10]. It is composed of two subsystems: the Hadoop runtime subsystem that executes the MapReduce applications and Hadoop distributed file system (HDFS) that handles data management and access [18].

The Hadoop runtime system includes one *JobTracker* and multiple *TaskTrackers*. The JobTracker receives submitted MapReduce jobs and schedules the tasks of each job onto the TaskTrackers. HDFS is composed of one *NameNode* and multiple *DataNodes*. The NameNode is responsible for the metadata management, while DataNodes store the data. Most configurations usually have dedicated centralized nodes to host the JobTracker and NameNode processes. Furthermore, each host/node in the system works as both TaskTracker and DataNode.

Files in HDFS are organized in equal-sized data blocks (e.g. 64MB). The data blocks are dispatched randomly onto the participating nodes for balanced data distribution. Each input data block corresponds to one map task (mapper). Under the design principle of data locality, each host first uses its best effort to run local tasks. A task is referred to as *straggler* if its progress is significantly slower than other tasks, delaying the entire MapReduce application [19]. If a node has completed all its local tasks and the MapReduce application is not completed yet, JobTracker will allocate stragglers to the idle node, which triggers data migration between nodes.

A host in the non-dedicated distributed computing environment could be interrupted randomly (with higher probability and frequency than typical failures), and potentially leave the system. As a result, the unavailability of the host will fail the tasks currently running on it and harm the overall MapReduce application performance. The node could return to the system after the interruption is serviced and could continue to participate in the MapReduce application. Data blocks are stored on persistent storage and could be reused after the node is back. However, the interrupted tasks have to be re-executed. An interrupted task can be re-executed on the same node if the task has not started executing on another node yet when the interrupted node with the corresponding data block returns to the system. The interrupted task could also be considered as a straggler, and be scheduled to another idle node, leading to non-trivial data migration.

III. AVAILABILITY-AWARE DATA PLACEMENT ALGORITHM

In this section, we first introduce a stochastic model to predict the execution time of each task under interruptions. Based on the model, an availability-aware data placement algorithm is proposed to build the foundation of ADAPT.

A. *Problem Statement*

We assume the inter-arrival times of interruptions for each host $i\ (0 \le i \le n-1)$ are independent and identically distributed (iid) as exponentially distributed random variables with parameter λ_i, which is the inverse of MTBI. We will use λ to replace λ_i when the context is clear.

The interruption service time (recovery time) follows a general distribution with mean μ. Interruptions may overlap with each other, meaning that an interruption could occur during the recovery of a previous interruption. In such cases, the interruptions are handled on First-Come-First-Serve (FCFS) basis. The latter interruption will be queued for recovery until all the previous interruptions are fixed. Each host in the system can be considered as an M/G/1 queuing system when we treat an interruption as an event (customer) in the queue [20].

B. *Performance of One Task*

In this subsection we give a stochastic model to predict the execution time of one task under aforementioned interruption patterns.

We term the failure-free execution time of one task as its length and represent it as γ. A task is completed successfully if there is no interruption occurring during length γ. If an interruption happens at time $t < \gamma$, we have to resume the task after the failure is recovered, which initializes another attempt to finish the task. Such attempts to a successful task execution continue until the termination condition is met.

Let random variable T denote the time to finish a task of length γ, we have

$$T = X_1 + X_2 + \ldots + X_S + Y_1 + Y_2 + \ldots + Y_S + \gamma$$
$$= \sum_{i=1}^{S} X_i + \sum_{i=1}^{S} Y_i + \gamma \qquad (1)$$

Here $X_i(1 \le i \le S)$ represents the rework cost and $Y_i(1 \le i \le S)$ is the system downtime due to one or more interruptions. S is the random variable denoting the number of interruptions occurred over the task execution. We use X and Y to replace X_i and Y_i when the context is clear. Figure 1 illustrates the composition of one task under interruptions.

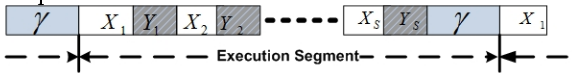


Figure 1: Task Execution under Interruptions

X falls within the range of $0 < t < \gamma$ and its probability of equaling to t is the conditional probability that an interruption happens at time t, given that the inter-arrival of interruption is less than γ. So we have the probability density function of X as $f_X(X=t) = \frac{\lambda e^{-\lambda t}}{1-e^{-\lambda t}}$.

The mean of X can be expressed as:

$$E(X) = \int_{t=0}^{\gamma} t f_X(X=t) = \frac{1}{\lambda} + \frac{\gamma}{1-e^{-\lambda t}} \qquad (2)$$

Under the assumption of M/G/1 interruption processing, we can get the mean of Y using existing results from queuing theory as,

$$E(Y) = \frac{\mu}{1-\lambda\mu} \qquad (3)$$

S is the number of failed attempts before a successful one, so we have the probability function of random variable S as $P(S=s) = (1-e^{-\gamma\lambda})^s e^{-\gamma\lambda}$. The first term of $(1-e^{-\gamma\lambda})^s$ is the probability of first failed s attempts due to interruptions. The second term of $e^{-\gamma\lambda}$ represents the probability of continuous failure-free segment execution. We can derive the mean and variance of S as:

$$E(S) = \frac{1-e^{-\gamma\lambda}}{e^{-\gamma\lambda}} = e^{\gamma\lambda} - 1 \qquad (4)$$

Put the formulas together, we have the mean of T as:

$$E(T) = E(E(T \mid S)) = E(\gamma + SE(X) + SE(Y))$$
$$= (e^{\gamma\lambda} - 1)(\frac{1}{\lambda} + \frac{\mu}{1-\lambda\mu}) \qquad (5)$$

C. *Availability-aware Data Placement Algorithm*

The availability heterogeneity leads to a non-uniform time to process data blocks for each node. Suppose a MapReduce application has m data blocks as input on a system with n nodes. The objective of the data placement algorithm is to find an optimized mapping from data blocks to the nodes, such that all nodes complete their assigned blocks at the same time (the pseudo code of the algorithm is presented in algorithm 1).

```
Subroutine buildHashTable //Build the hash table
Input: The measured interruption arrival rate $\lambda$, interruption service time
$\mu$, and the failure-free execution time of a task $\gamma$. We have $m$ data
blocks as input on a system with $n$ nodes.
Output: Hash table $HT[0...m-1]$
BEGIN
FOR EACH node $i\ (0 \le i < n)$
    Calculate its expected task execution time $E_i(T)$ based on formula (5).
END FOR
$\Phi = \sum_{i=0}^{i<n} \frac{1}{E_i(T)}$
$a = 0$ //begin index of hash table keys for node $i$
FOR EACH node $i\ (0 \le i < n)$
    $rate_i = \frac{1/E_i(T)}{\Phi}$
    $w_i = m \times rate_i$ //end index of hash table keys for node $i$
    $b = \lceil a + w_i \rceil$
    FOR $\lfloor a \rfloor \le j < \lceil b \rceil$
        insert i to list $HT[j]$;
    END
    $a = b$
END FOR
END

Subroutine dataPlacement //Data block placement.
input: hash table $HT$ generated from buildHashTable
output: id of the node that hosts the data block
BEGIN
Generate random integer r such that $r\ (0 \le r < m)$
IF $size(HT[r]) = 1$
  return $HT[r]$
ELSE //handle the collisions.
    $\Omega = \sum_{k=0}^{k<size(HT[r])} rate_k$
    generate random number $r_1$ such that $(0 \le r_1 < 1)$
    lowBound=0;
    FOR EACH $j \in HT[r]$
      upbound=lowbound+ $rate_i / \Omega$
      IF lowbound $\le r_1 <$ upbound
          return $j$
      END IF
      lowbound=upbound
    END FOR
END IF
END
```

Algorithm 1: Availability-aware Data Placement Algorithm

The traditional data placement algorithm randomly selects nodes for each data block. In particular, when a data block arrives, the NameNode generates a random integer $r\ (0 \le r < n)$ and selects the corresponding data node with index r to hold the block. The availability-aware data placement also utilizes random number to make the decision but improves the existing approach by differentiating the weight of each host to dispatch data blocks.

We have two subroutines, *buildHashTable* and *dataPlacement*. The buildHashTable subroutine builds a hash table that maps data blocks to nodes. We first calculate the expected task execution time of each node under interruptions based on formula (5). The number of blocks allocated to each node should be proportional to its efficiency of processing data blocks, which is reflected by $1/E_i(T)$. We next calculate the number of data blocks that should be allocated to node i, which is denoted as w_i. The hash table uses the block id as the key and the corresponding node index as the value. In case of hash table *collisions*, we use a chain to deal with multiple values mapping to the same key. The hash table is created every time when the MapReduce application initializes its input in the distributed file system.

Subroutine *dataPlacement* makes the data placement decision. We first generate a random integer $r\ (0 \le r < m)$, which is used as the key to locate items in the hash table HT. If there is only one value corresponding to the key of r, we simply select the corresponding node to host the data. In case of collisions on the key, we will generate another random number between 0 and 1, and iterate each value in the collision chain to select the host.

It is noteworthy to mention that the availability-aware data placement algorithm is a superset of the existing approach. It is logically equivalent to the existing data placement algorithm if all the nodes share the same availability pattern.

IV. ADAPT DESIGN, IMPLEMENTATION AND METHODOLGIES

In this section, we introduce the design and implementation of ADAPT in Hadoop. Other design tradeoffs are also discussed.

A. *ADAPT Design*

Figure 2 overviews the architecture of ADAPT. Basically, ADAPT modifies the NameNode and Client of the existing MapReduce framework. ADAPT adds a *Performance Predictor* component to the NameNode. The performance predictor retrieves the parameters of interruption arrival rate, interruption service time of each data node from the heart beat collector. The failure free execution time of the Map task is obtained from the logging services of Hadoop. Based on the input and formula (5), the performance predictor is able to deliver the expected execution time of Map tasks for each node.

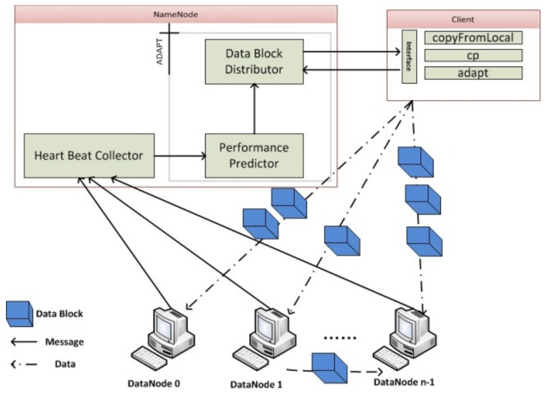


Figure 2: Overview of ADAPT

Data Block Distributor is the component that implements the data placement algorithm. The existing data placement strategy does not differentiate the availability of the nodes. ADAPT improves the existing approach by implementing algorithm 1, which considers the availability pattern as a factor in making the data placement decision.

Currently we have three interfaces to utilize ADAPT, namely *copyFromLocal, cp,* and *adapt.* The first two interfaces are natively supported by the HDFS shell and are modified to implement the ADAPT feature. The last interface *adapt* is a newly added Hadoop shell command that helps to reorganize the data placement of the existing files in HDFS. *copyFromLocal* is used to govern the data copied from local file systems to HDFS using ADAPT. Interfaces such as *cp* and *adapt* are used to reshape the data distribution on HDFS to be availability aware.

B. *ADAPT Implementation*

1) *ADAPT on the NameNode Process*

MapReduce currently adopts centralized metadata management mechanism. The file system metadata information is held in the memory of NameNode. ADAPT limits its footprint on the NameNode. To conserve resource, the NameNode does not record the history heart beat information of all the nodes. Instead, a data structure with two double data types is used to save the interruption arrival rate and recovery time for each node. The data structure is updated whenever the heart beat arrivals/misses are sufficient to change its values.

As shown in Figure 2, the Performance Predictor is implemented as a method of NameNode. When ADAPT functionality is invoked by the client and received by the NameNode, a hash table is created with the mapping information from data blocks to nodes. The size of the hash table is equivalent to the number of blocks. We note that the lifetime of the hash table is relatively short, as it is created when ADAPT is called by the client, and deleted when the corresponding data blocks have been distributed.

2) *ADAPT on Client Processes*

We add another argument to HDFS shell commands of *copyFromLocal* and *cp*, which is used as a flag to enable or disable ADAPT. HDFS can be configured and used in its original implementation, if ADAPT is disabled. The shell command of *adapt* takes a file name as input, and redistributes the data blocks of the file to become availability aware. The implementation of *adapt* shell command is analogous to that of *rebalance* functionality, which is natively supported by Hadoop.

C. *ADAPT Analysis*

The existing random data placement strategy of MapReduce balances the data distribution on each node. Nevertheless, ADAPT is designed to achieve another dimension of balance: guarantee that all the nodes complete the processing of their allocated blocks at the same time. We discuss the design tradeoffs of the two balance schemes as follows.

The first concern of ADAPT is that it potentially increases the data transfer cost when copying data from local file system to HDFS using *copyFromLocal*. The more reliable nodes have more data blocks to transfer and could delay the entire data migration between local file system and HDFS.

We note that hosts could also be interrupted during the data transfer phase, and ADAPT could help to reduce the cost as data blocks are distributed taking into consideration node availabilities. Based on our experience, we believe the impact of interruptions on the job execution time is typically more significant than that of data transfer cost. Interruptions during job execution not only impact HDFS, but also the MapReduce runtime system. The system is more complex for job execution than data transfer, which makes it more vulnerable to interruptions. The performance impact is further exasperated by deadline-driven jobs.

Another concern about ADAPT is that it still leads to more disk space consumption for more reliable nodes, which might perhaps be an unwanted feature by users. To address this problem, we set a threshold for the number of blocks that can be allocated on each node. The node that reaches the threshold will not be considered for future data block placement. This solution helps to tune the data placement and maintain the user fidelity. Suppose a MapReduce has m blocks and we have n nodes in the system, k is the number of replicas. We set the threshold as $m \times (k+1)/n$ such that the data blocks allocated to each node do not exceed its expected number with one more replica.

ADAPT deals with the input data distribution and directly optimizes the performance of the map phase. There is no immediate relationship between the data placement strategy and the reduce phase. We target at improving the map phase cost in this study, and leave the reduce phase optimization for future work.

V. PERFORMANCE EVALUATION

We have prototyped ADAPT in the Hadoop framework and conducted experiments in an emulated non-dedicated distributed computing environment to evaluate the performance. To validate the potential of ADAPT for large-scale computing environment, we have also performed large-scale simulations on failure traces from production systems.

A. Experiment Settings

The experiments were conducted on Magellan [11], a cloud computing platform from Argonne National Laboratory. Magellan is composed of 504 compute nodes, each with dual 2.66 GHz Intel Quad-core Nehalem processors, 24GB of memory, and 500GB SATA hard drive. We have created 32 to 256 virtual machine (VM) instances on Magellan to emulate the non-dedicated distributed computing environment. Each VM instance is allocated with one CPU core, 2GB RAM and 10GB storage space [21]. We installed Ubuntu 9.04 with kernel 2.6.28.10 on all the nodes. Each node has about 5GB to store MapReduce data blocks, in addition to the OS and system-level software.

We have prototyped ADAPT using Hadoop 0.20.2, and deployed the prototype on the emulated environment. We limited the network bandwidth from 4Mb/s to 32Mb/s to simulate network bandwidth in distributed environment. We also injected interruptions to the TaskTracker and DataNode processes of each node to emulate the volatility of the non-dedicated environment. By default half of the nodes in the system were interrupted. To enforce heterogeneity in availability, the interrupted nodes were further divided evenly into four groups with different availability patterns. Table 2 summarizes the MTBI and the mean interruption service time of each group. Interruptions are injected based on the assumed distributions.

Table 2: Overview of the Interrupted Nodes

	MTBI (seconds)	Service Time (seconds)
Group 1	10	4
Group 2	10	8
Group 3	20	4
Group 4	20	8

Terasort was used as the benchmark [22]. The block size was set to the default 64MB and the number of data blocks was set proportional to the number of nodes such that each node had 20 blocks on average (about 1.28 GB of data). We examined the performance of replication with both 1 and 2 replicas.

We varied three parameters in the experiments, the ratio of interrupted nodes, bandwidth and the total number of nodes. The time consumed by the map phase was measured as the performance metric. Data locality is defined as the ratio of local tasks to all tasks. It was also measured for a comprehensive performance analysis. A higher data locality typically means less data migration cost as well as less network traffic. The elapsed time and data locality are presented in figure 3 and 4, respectively. We had 10 runs for each scenario and derived their means for performance evaluation[1]. Table 3 summaries the default values of the parameters used in the experiment.

[1] We observed some unexpected bad performance in the experiments. A detailed analysis revealed that the anomalies were due to the VM contentions on the same physical machine. We discarded such data since the VM contention problem is out of the scope of this study. The collected data presents moderate variances by removing the abnormal data.

Table 3: Default Values in the Experiment

Parameter	Value
Block Size	64MB
Ratio of Interrupted Nodes	1/2
Bandwidth	8Mb/s
Number of Nodes	128

B. Experiment Results

1) Performance with Different Number of Interrupted Nodes

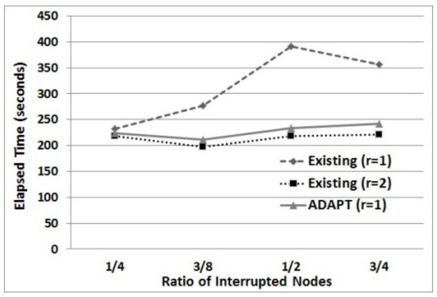
We varied the ratio of interrupted nodes from 1/4 to 3/4. The values of other parameters were set as the default values specified in table 2 and 3. Figure 3(a) and 4(a) report the application elapsed time and the corresponding data locality. When the level of replication is set to 1 replica, ADAPT outperforms the existing approach significantly. Particularly, if half of the nodes are interrupted, ADAPT has an elapsed time of 234 seconds, which improves the existing approach of 391 seconds by 40%.

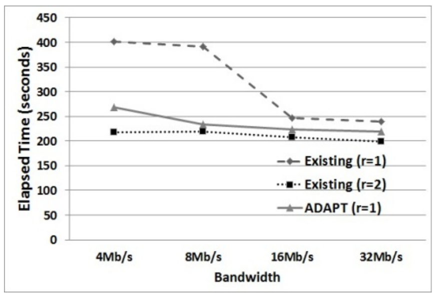
If we compare the performance of ADAPT with one replica with traditional Hadoop with 2 replicas, we observe slightly worse performance. However, the degradation is acceptable considering the advantage of ADAPT in storage space efficiency. The existing approach does not consider host heterogeneity, incurs significant data migration between the nodes with different reliabilities, and leads to the lowest data locality as shown in figure 4(a). ADAPT presents stable data locality regardless of the interrupted nodes ratio.

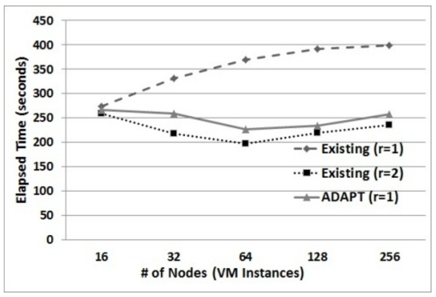
An interesting observation is that the elapsed time of traditional Hadoop with one replica has better performance when 3/4 nodes are interrupted, compared with the case of 1/2 interrupted nodes. A detailed study reveals that the system has the highest availability variance when 1/2 nodes are interrupted, which leads to more data migration overhead, and eventually slows down the performance. Figure 4(a) shows that the traditional Hadoop with 1 replica has a significantly lower data locality (e.g. 87%) when half the nodes are interrupted.

2) Performance with Different Bandwidth

We varied the network bandwidth from 4Mb/s to 32Mb/s, set other parameters as the default values specified in table 2 and 3, and plotted the performance in figure 3(b) and 4(b). Although ADAPT shows a performance advantage over the existing approach, its benefit decreases as the network bandwidth goes up. Due to the relatively lower data migration cost, nodes can run more remote tasks when the network bandwidth is higher, which helps to mitigate the impact of failures. We can observe from figure 4(b) that the existing approach with 1 replica has a lower data locality when the bandwidth is 8Mb/s or higher. The existing







(a) Different Ratio of Interrupted Nodes (b) Different Network Bandwidth (c) Different # of Nodes

Figure 3: Elapsed Time

approach tolerates failures at the sacrifice of lowering data locality. ADAPT keeps a high data locality in figure 4(b) while getting satisfactory performance improvement in figure 3(b).

Although the benefit of ADAPT is less significant for higher bandwidth, we still observe a constant advantage of data locality in figure 4(b). This observation confirms the potential ADAPT in fostering high data locality and reducing the network traffic for heterogeneous computing environment with better network connectivity.

3) *Performance with Different Number of Nodes*

Figure 3(c) and 4(c) depict the performance with different number of nodes and show the potential of ADAPT in term of scalability. Other parameters are set with the values specified in table 2 and 3. We observe considerable performance degradation for the existing approach with 1 replica while the system scales up. This reveals that the performance of the existing approach is not scalable under interruptions and has a limitation in enabling large-scale computing environment. On the other hand, ADAPT presents relatively stable performance across all system sizes, comparable to traditional Hadoop with 2 replicas.

C. *Simulation Results for Large-Scale Distributed Computing Environments*

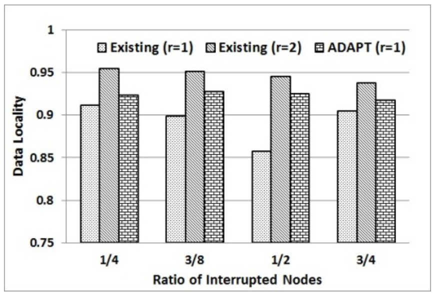
We have shown the performance gain of ADAPT in an emulated non-dedicated distributed computing environment in subsection V-B. The scale with 256 nodes or less in the emulated environment has several limitations. For example, we have to inject frequent interruptions to make their impact visible. The case of 3 replicas is also omitted in the experiments since it showed little difference with the 2 replica scenario. It is also not clear if the distribution assumed by the models work well in practice.

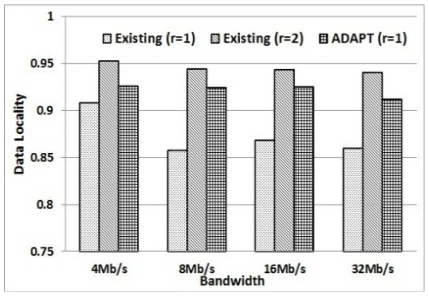
To address the aforementioned concerns, we have performed simulations in a large-scale distributed computing environment. Based on the source code analysis of Hadoop, a discrete event simulator was developed with mechanism analogous to that of Hadoop. The simulator injects failures based on the data collected from SETI@home [9]. The failure traces were collected from 226208 nodes in 1.5 years from the nodes that participated in SETI@home. We randomly selected 16384 nodes to build the simulated non-dedicated distributed environment.

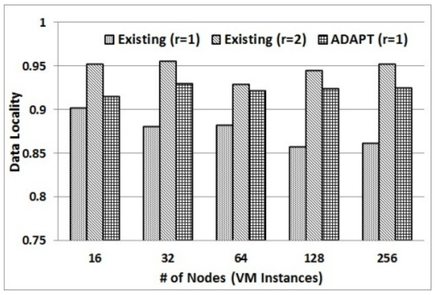
For an in-depth understanding of the performance, we have measured the overhead of each component (e.g. rework, recovery, migration and misc). The *misc* overhead refers to cost in addition to the other three components. It includes the additional scheduling cost, the cost of duplicated straggler execution, and the idle time at the end of the map phase. We used the aggregated failure-free execution time of the application as the base performance and derived the overhead ratio of each component.

A straightforward alternative to ADAPT is to dispatch the data blocks based on the availability of each node, $\frac{MTBI - \mu}{MTBI}$. We also implemented this naïve data placement strategy and measured the overhead for a comprehensive performance evaluation.

We presented the impact of the network bandwidth, the block size, and the number of nodes in figure 5. In each subfigure of figure 5, we varied one parameter and

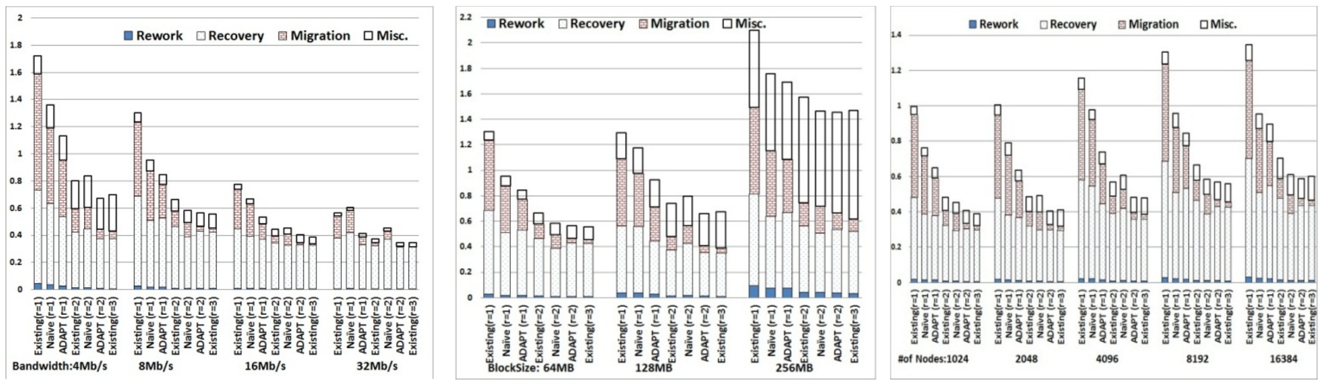






(a) Different Ratio of Interrupted Nodes (b) Different Network Bandwidth (c) Different # of Nodes

Figure 4: Locality



(a) Different Network Bandwidth (b) Different Block Size (c) Different # of Nodes

Figure 5: Simulation Results

fix the other parameters at the default values as listed in table 4.

Figure 5(a) demonstrates the overhead ratio by varying the network bandwidth from 4Mb/s to 32Mb/s. The impact of interruptions is more significant for fewer replicas with lower network bandwidth. Particularly, the existing approach of 1 replica incurs an overhead of 172% to the failure-free elapsed time. ADAPT presents considerable performance gain for lower network bandwidth with less number of replicas. It helps to reduce both the migration and recovery cost. ADAPT constantly saves the migration cost by half or more for all the scenarios. ADAPT with 2 replicas has a performance comparable to the existing approach with 3 replicas, if not better than. The naïve data placement strategy outperforms the traditional approach, but still performs worse than ADAPT.

Table 4: Default Values in the Simulation

Parameter	Value
Network Bandwidth	8Mb/s
Block Size	64MB
Number of Nodes	8196
Average Number of Tasks per Node	100
Failure-free Task Execution Time (64MB data block)	12s

Figure 5(b) reports the performance with different block sizes. A larger block size usually results in considerable performance degradation and is not recommended in a failure-prone environment. We observe that the overhead of each component rises at different degrees while increasing the block size. Misc overhead dominates the performance for larger blocks size.

Our detailed study reveals that it is because a larger block size usually leads to more duplicated straggler execution cost and idle waiting time. ADAPT works well to optimize the data migration cost for large block size but helps little to benefit the overall performance. This is mainly due to the fact that migration overhead is not the dominant factor in the overall performance degradation.

We varied the number of nodes from 1024 to 16384 and presented their performance in figure 5(c). Interruptions introduce more overhead as the system scales up. In particular, we observed an increasing migration and recovery cost overhead for more number of nodes. ADAPT helps to reduce the migration cost by at least 50% for all the scenarios. ADAPT with 2 replicas has a performance close to the existing approach with 3 replicas, and hence delivers the same levels of performance with significantly improved storage space efficiency.

VI. RELATED WORK

Volatility has constantly been recognized as a major concern faced by the non-dedicated distributed computing environment. Previous studies mainly focused on the performance modeling and optimization from the perspective of job scheduling and task allocation [7] [8] [23] [24]. We differentiate our work with these studies by setting our research in the context of the emerging MapReduce parallel computing paradigm. Our study is unique due to the granularity (e.g. task-level) and our focus on the data-placement strategies. We adopted the granularity of the performance modeling to be one task, rather than the entire workloads as adopted by the existing studies [8]. Also, current studies have mostly focused on managing compute resources, and not on the data placement. ADAPT optimizes the performance from the perspective of data placement, rather than task allocation as addressed by the exiting works.

In [6], the authors proposed MOON to optimize the performance of MapReduce in distributed opportunistic environment. The basic idea of MOON is to place a small portion of dedicated servers in the system to enhance the reliability. ADAPT targets an Internet based distributed computing environment and intends to optimize the performance of MapReduce applications without the need of extra dedicated servers. The availability-aware data placement strategy can also benefit MOON by considering the dedicated nodes as ultra–reliable and differentiating the data distribution.

In [19], Zaharia et. al. recognized the performance limitations on the speculative task execution for stragglers and proposed LATE (Longest Approximation Time to End) algorithm to improve the response time of MapReduce applications. We believe scheduling algorithms are important, but are an orthogonal research problem. There is a performance improvement space by developing availability-aware MapReduce scheduling algorithms, in order deliver higher QoS guarantees to the applications. We plan to pursue this avenue in our future work.

In [25], the authors proposed to improve MapReduce under heterogeneous cluster environment via data placement. We extend [25] by considering the heterogeneity of reliability in non-dedicated distributed computing environment. The impact of reliability on MapReduce performance is implicit and cannot be measured directly as done by [25]. One contribution of this work is that we build a stochastic model to unveil the impact of reliability and incorporate the model into the MapReduce framework.

Efforts have been made to build aggregated storage systems based on the idle storage space in distributed environment [26] [27] [28] [29]. In [26], the authors proposed a storage architecture that includes both volatile and durable nodes to balance between reliability and performance. Nguyen et. al. proposed differentiated replication strategy for data centers with the consideration of data locality [29]. All of these systems focused on the optimization of storage systems (e.g. data durability), however, ADAPT differs from these works by using end-to-end application performance as the guideline for performance optimization

A recent study by Palanisamy et. al. [30] presented Purlieus, a MapReduce resource allocation system aimed at enhancing the performance of MapReduce jobs in the cloud. Purlieus provisions virtual MapReduce clusters in a locality-aware manner enabling MapReduce virtual machines (VMs) access to input data and intermediate data from local or close-by physical machines. Purlieus proposed a data placement strategy that considers both the job characteristics and the workload of the physical machines. ADAPT optimizes MapReduce in non-dedicated distributed environment, instead of the cloud environment as addressed by Purlieus, which inherently has different performance and availability characteristics. While Purlieus can be adopted to improve the locality of multiple jobs, combining ADAPT and Purlieus could complement each other to improve MapReduce with multiple jobs in non-dedicated distributed environments.

VII. CONCLUSIONS AND FUTURE WORK

The computing power distributed in the Internet has true potential to solve technical challenges. However, traditional parallel programming models assume tightly coupled and reliable computing platforms that mismatch the features of Internet based computing. Such gap considerably limits the applications suitable for the non-dedicated distributed computing environments. MapReduce has emerged as a compelling programming paradigm due to its easy programming, automatic parallelism and the support of fault tolerance. With the help of the presented work, we have shown how MapReduce can be used to harness non-dedicated computing and support a larger variety of applications. The inherit volatility of non-dedicated computing continues to challenge MapReduce applications. The frequent data migration and the network bandwidth limitations further amplify the problem.

In this paper, we proposed ADAPT, an availability-aware MapReduce data Placement strategy to optimize the performance of MapReduce applications, increasing the data locality and reducing the network traffic. We first proposed an analytical model to estimate the failure-prone performance of MapReduce tasks, which dynamically steers the availability-aware data placement algorithm. We introduced the design of ADAPT and its implementation on Hadoop, an open-source implementation of MapReduce. ADAPT incurs little overhead to the maintenance of the centralized NameNode. Furthermore, ADAPT is implemented as an optional feature of Hadoop and can be easily disabled with no impact on the original Hadoop.

We have conducted extensive experiments and simulations to evaluate the performance of ADAPT. The experiments were performed on an emulated non-dedicated distributed computing environment on Magellan, a cloud computing platform at Argonne National Laboratory. We also carried out simulations with failure traces of production systems to evaluate the potential of ADAPT for large-scale computing environments. The experimental and simulation results have demonstrated that ADAPT helps to reduce the application runtime by more than 30%. It provides comparable performance with less replication, yielding improved storage efficiency. ADAPT significantly increases the data locality and reduces the data migration cost. Although the advantage of ADAPT in improving MapReduce application performance is less significant for environment with higher network connectivity, it still helps to improve the data locality and reduces the network traffic, which are considered as important performance metrics by recent studies [25] [30] [31].

As future work, we plan to develop an availability-aware MapReduce job scheduling strategy and optimize the reduce phase performance. The ultimate goal of our research is to build a failure-agnostic framework to support MapReduce under non-dedicated distributed computing environments, while improving storage efficiency.

ACKNOWLEDGMENT

This research was supported in part by National Science Foundation under NSF grant CCF-0621435, CCF-0937877 and CNS-0751200. We are thankful to the Magellan project of Argonne Leadership Computing Facility at Argonne National Laboratory for the support of running the experiments. We thank Adam Scovel from Argonne Leadership Computing Facility for his assistance with the experiments execution.

REFERENCES

[1] SETI@HOME Website. [Online]. http://setiathome.berkeley.edu/
[2] Folding@HOME Website. [Online]. http://folding.stanford.edu/

[3] D. Thain, T. Tannenbaum, and M. Livny, "Distributed Computing in Practice: The Condor Experience," *Concurrency and Computation: Practice and Experience*, 2004.

[4] MPI Website. [Online]. http://www.mcs.anl.gov/research/projects/mpi/

[5] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM - 50th anniversary issue: 1958-2008*, vol. 51, no. 1, 2008.

[6] H. Lin, X. Ma, J. Archuleta, W. Feng, M. Gardner, Z. Zhang, "MOON: MapReduce On Opporutunistic eNvironments," in *Proc. of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC)*, 2010.

[7] R. Raman, M. Livny, M. Solomon, "Matchmaking: Distributed Resource Management for High Throughput Computing," in *Proc. of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC)*, 1998.

[8] M. Wu and X. -H. Sun, "Grid Harvest Service: A Performance System of Grid Computing," *Journal of Parallel and Distributed Computing*, vol. 66, no. 10, pp. 1322-1337, 2006.

[9] FAILURE TRACE ARCHIVE. [Online]. http://fta.inria.fr/apache2-default/pmwiki/index.php?n=Main.Purpose

[10] Hadoop Website. [Online]. http://hadoop.apache.org/

[11] Magellan: a cloud for science. [Online]. http://magellan.alcf.anl.gov/

[12] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, "The Google File System ," in *Proc. of 19th ACM Symposium on Operating Systems Principles*, 2003.

[13] Cheng T. Chu, Sang K. Kim, Yi A. Lin, Yuanyuan Yu, Gary R. Bradski, Andrew Y. Ng, Kunle Olukotun, "Map-Reduce for Machine Learning on Multicore," in *Proc. of Neural Information Processing Systems (NIPS)*, 2006.

[14] T. Tu, C. A. Rendleman, D. W. Borhani, R. O. Dror, J. Gullingsrud, M. O. Jensen, J. L. Klepeis, P. Maragakis, P. Miller, K. A. Stafford, and D. E. Shaw, "A Scalable Parallel Framework for Analyzing Terascale Molecular Dynamics Simulation Trajectories," in *Proc. of the ACM/IEEE Conference on Supercomputing*, 2008.

[15] J. Ekanayake, S. Pallickara, and G. Fox, "MapReduce for Data Intensive Scientific Analyses," in *Proc. 4th IEEE International Conference on eScience*, 2008.

[16] Y. Kwon, D. Nunley, J. P. Gardner, M. Balazinska, B. Howe, and S. Loebman., "Scalable Clustering Algorithm for N-Body Simulations in a Shared-Nothing Cluster," in *Proc. of SSDBM*, 2010.

[17] Dryad Project Website. [Online]. http://research.microsoft.com/en-us/projects/dryad/

[18] Hadoop Distribute Filesystem Website. [Online]. http://hadoop.apache.org/hdfs/

[19] M. Zaharia, A. Konwinski, A.D. Joseph, R. Katz and I. Stoica, "Improving MapReduce Performance in Heterogeneous Environments," in *Proc. of 8th USENIX Symposium on Operating System Design and Implementation (OSDI)*, 2008.

[20] D. Gross, C. M. Harris, *Fundamentals of Queuing Theory*, 3rd ed.: John Wiley Sons, 1998.

[21] Lavanya Ramakrishnan, Piotr T. Zbiegel, Scott Campbell, Rick Bradshaw, Richard Shane Canon, Susan Coghlan, Iwona Sakrejda, Narayan Desai, Tina Declerck, Anping Liu, "Magellan: experiences from a science cloud," in *Proc. of the 2nd international workshop on Scientific cloud computing (ScienceCloud)*, 2011.

[22] Package terasort of Hadoop. [Online]. http://hadoop.apache.org/common/docs/current/api/org/apache/hadoop/examples/terasort/package-summary.html

[23] Berman, F.; Wolski, R.; Casanova, H.; Cirne, W.; Dail, H.; Faerman, M.; Figueira, S., etc, "Adaptive computing on the Grid using AppLeS," *IEEE Trans. on Parallel and Distributed Systems*, vol. 14, no. 4, pp. 369 - 382 , 2003.

[24] Fran Berman , Rich Wolski , Silvia Figueria , Jennifer Schopf , Gray Shao , Silvia Figueira Jennifer Schopf , "Application-Level Scheduling on Distributed Heterogeneous Networks," in *Proc. of Supercomputing*, 1996.

[25] J. Xie, S. Yin, X.-J. Ruan, Z.-Y. Ding, Y. Tian, J. Majors, and X. Qin, "Improving MapReduce Performance via Data Placement in Heterogeneous Hadoop Clusters," in *Proc. 19th Int'l Heterogeneity in Computing Workshop (HCW'10)*, 2010.

[26] Abdullah Gharaibeh, Samer Al-Kiswany, Matei Ripeanu, "ThriftStore: Finessing Reliability Tradeoffs in Replicated Storage Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 6, 2011.

[27] Atul Adya, William J. Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R. Douceur, Jon Howell, Jacob R. Lorch, Marvin Theimer, and Roger P. Wattenhofer, "FARSITE: Federated, available, and reliable storage for an incompletely trusted environment," in *Proc.s of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*, 2002.

[28] S. Vazhkudai, X. Ma, V. Freeh, J. Strickland, N. Tammineedi, and S. Scott, "Freeloader: Scavenging desktop storage resources for bulk, transient data," in *Proc. ACM/IEEE Conference on Supercomputing*, 2005.

[29] Tung Nguyen, Anthony Cutway and Weisong Shi, "Differentiated Replication Strategy in Data Centers," in *Proc. of the 7th IFIP International Conference on Network and Parallel Computing (NPC)*, 2010.

[30] Balaji Palanisamy, Aameek Singh, Ling Liu, Bhushan Jain, "Purlieus: Locality-aware Resource Allocation for MapReduce in a Cloud," in *Proc. of ACM/IEEE Conference on SuperComputing*, 2011.

[31] Yandong Wang, Xinyu Que, Weikuan Yu, Dror Goldenberg, Dhiraj Sehgal, "Hadoop Acceleration through Network Levitated Merging," in *Proc. of ACM/IEEE Conference on SuperComputing*, 2011.