MapReduce on Opportunistic Resources Leveraging Resource Availability

Kyungyong Lee
ACIS Lab. Department of ECE
University of Florida
klee@acis.ufl.edu

Renato Figueiredo
ACIS Lab. Department of ECE
University of Florida
renato@acis.ufl.edu

***Abstract*—MapReduce is a popular large-scale parallel data processing framework. In the context of MapReduce processing on volunteer computing environments, it is important to devise scheduling and data placement policies that account for characteristics of opportunistic resources. This paper investigates availability characteristics of opportunistic resources with analyses based on log traces from the SETI@Home project. Based on the analysis, the paper devises heuristics to leverage the uptime of each available session to detect possibly long-lasting resources. Our proposed session uptime-based resource availability prediction approach shows a two-fold reduction in the number of service disturbance compared to an availability-rate based model. The paper paper investigates a heuristic that differentiates stable nodes from unstable nodes while increasing the chance of leveraging existing data blocks.**

***Keywords*-Opportunistic computing, volunteer computing, MapReduce, Hadoop, uptime**

I. Introduction

Large-scale distributed systems have become mainstream platforms to address challenges encountered in a wide variety of compute- and data-intensive applications. Among such platforms are Volunteer Computing (VC) systems — desktop "Grids" that scavenge idle computing power of Internet-connected commodity computers. Resources in VC systems embrace characteristics of not only heterogeneous configurations of hardware, software, and network, but also wide geographical distribution, allowing them to potentially reach very large number of candidate resources in the world. However, given the characteristics of VC systems, application scenarios are typically restricted to those that map well to independent computation across large number of resources, such as Bag-of-Tasks and parameter sweeping.

Google MapReduce [1] and Hadoop [2] are widely used for large-scale data processing problems, such as counting the URL access frequency and reverse Web-link graph analysis for Internet-scale graphs. They borrow the concept of Map and Reduce from functional programming languages, while enabling parallel execution of data processing jobs on large-scale computing environments and freeing users from job distributions and handling resource failures. Despite of their popularity, Map/Reduce platforms are deployed mostly on dedicated high performance data centers that are available exclusively to a company or a research institution, and such environments can suffer from the dearth of available resources.

There are benefits in using VC systems to support MapReduce type applications. First, VC can contribute a large number of compute and storage resources to a MapReduce platform. For example, the flagship VC project SETI@Home provides 28.8 PBytes of disk storage aggregated from about 500,000 machines during the month of August/2010 [3]. In comparison, large-scale dedicated MapReduce systems at Yahoo, Facebook and eBay are estimated to have of the order of thousands of nodes with tens of PBytes in storage [4]. The large number of publicly-available donated resources can help to build opportunistic MapReduce systems that provide massive capacity for scientists who do not own dedicated resources; furthermore, the simple MapReduce programming model and the ability of the runtime system to handle replicated data placement, data affinity in task placement, and transparently handle failures can enable wider use of VC systems for large-scale data-parallel applications.

However, the deployment of MapReduce on VC systems introduces several challenges. In general, the inherent resource volatility of VC can restrict availability of the MapReduce service. There are also implementation-specific issues; for instance, SETI@Home has about 100,000 active resources [5] at a time, and the Hadoop implementation can have scalability limitation at this scale [6]. Thus, heuristics to select a subset of VC resources for differentiated role assignment needs to be incorporated. In a Wide Area Network (WAN) environment, where most of VC resources are located, the limited end-to-end connection due to private IP addresses, network address translation (NAT), firewall devices among donated resources, and bandwidth limitations can limit correct operation of block replication and task execution in MapReduce frameworks.

In this paper, we address the challenge of providing stable MapReduce services on VC platform by leveraging the availability characteristics of volatile resources. The overall approach is to select a subset of nodes that are predicted to be available in the future. We leverage an overlay virtual network to recover end-to-end network connectivity in a WAN environment constrained by NATs and firewalls. Based on the thorough analysis of SETI@Home resource availability logs provided by the *Failure Trace Archive* [5], we confirmed that an available session with a longer uptime tends to stay available longer, and that distribution can be expressed as a Weibull function. Based on this observation we propose to select a subset of nodes with longer session-time to designate them as *datanodes* and *task-trackers*. In addition to using the longer uptime nodes for data placement and task execution, we propose the classification of nodes into groups named *regular-pool* and *infant-pool*. Resources in an *infant pool* contain valid data-blocks but the uptime of current session is short to satisfy the criteria to become a member of *regular-pool*. Nodes in a *regular-pool* satisfy the current session uptime length criteria, and they perform as ordinary *datanode* and *tasktracker* in Hadoop. *Namenode* and *jobtracker* prefer nodes in a regular-pool for block-placement and task scheduling, and nodes in a *infant pool* are selected for job execution when task-trackers in a regular-pool cannot handle given job request loads.

Simulations are conducted to measure the number of interruptions during MapReduce operations by replaying the availability log of SETI@Home [5]. The simulation results demonstrate that our proposed uptime-based resource selection mechanism provides more reliable service by reducing the number of interruptions in half compared to an availability-rate based mechanism [7] and by a factor of 12 compared to a random node selection mechanism.

In order to validate the proposed methods, we implemented the *infant-pool* block placement and task scheduling heuristic on top of the Hadoop default job-queue-task-scheduler and delay-scheduler [8]. We have evaluated the proposed heuristic on a real-world WAN cloud computing testbed, FutureGrid [9], by replaying the SETI@Home availability logs to capture the volatility of resources in a controlled manner. To evaluate the system in a realistic job submission scenario, we leverage SWIM [10], a statistical workload injector for MapReduce based on log files from Facebook. The experiment results demonstrate that the adoption of *infant-pool* improves the fault-tolerance of the system by decreasing the number of interrupted tasks about 40% when around 50% of nodes issue interruption event at least once during the experiments.

In summary, the research contributions of this paper are as follow:

- A novel uptime-based resource selection heuristic that provides more reliable and consistent service than the availability-rate based method.
- A novel *infant-pool* heuristic that can differentiate roles of VC resources based on their predicted availability.
- Evaluations of proposed methods on an emulated volatile environment replaying failure trace logs with Hadoop job submission scenarios by Facebook.

II. MapReduce on Volunteer Computing and Related Works

In this section, we describe the characteristics of VC resources that need to be considered to support MapReduce applications and the corresponding related work in the literature.

Volatility of Volunteered Resources
Participating nodes in a VC system can cease to donate their computing power at any time, which prevents the delivery of a reliable set of nodes to a MapReduce framework. In order to deal with this issue, MOON [11] proposes to leverage a small number of dedicated resources to provide reliable services, in combination with adaptive task and data scheduling algorithms. Similar to MOON, SpeQulos [12] proposes a mechanism to provide reliable services in an opportunistic computing environment by supplementing a small number of dedicated cloud-computing resources for redundant task execution at the end of job completion. Though they can improve reliability of a system with opportunistic resources, the proposed algorithms require additional dedicated resources where such nodes might not exist. In our proposed method, without relying on dedicated nodes, we leverage opportunistic resources that are likely to provide reliable service in the future based on the uptime of the current available session. Thus, our proposed heuristic can help approaches such as MOON and SpeQulos to decrease the number of dedicated resources that are needed.

Tang et. al. [13] presented an implementation of MapReduce on a VC environment. In order to improve fault-resilience, they leverage Bitdew [14], a data management and distribution framework on a distributed environment with volatile resources. They do not take into account the different degrees of availability among opportunistic resources. Our proposed uptime-based availability prediction method can contribute to improve the reliability of such a system.

Dynamic MapReduce Pool Configuration
The client/server-based architecture of MapReduce can impose scalability limitations [6], which requires careful monitoring in a VC environment where large numbers of available resources may overwhelm the MapReduce central manager. In order to maintain MapReduce pool size within a controllable number of resources based on job demands, an approach such as PonD [15] can be followed. Though dynamic resource pool size configuration is possible, we need to select a subset of volunteer resources that are likely to provide reliable services in a MapReduce pool. Kondo et. al. [16] and Javadi et. al. [17] analyzed availability logs of real-world opportunistic resources and proposed heuristics to predict highly-available nodes in the future based on the pattern of past availability.

Similar to those works, ADAPT [7] used availability history to predict future availability, and differentiated roles of participating nodes based on the expected availability on MapReduce. In this paper, we propose a different availability prediction method that relies on the uptime of a current session. Based on analysis of a SETI@Home log file, the historical availability-based prediction incurs twice more interruptions than our proposed uptime-based prediction mechanism.

Operation in a Wide-Area-Network Different from a data center network, VC resources are generally distributed across a WAN where end-to-end network connection is not guaranteed due to private IP addresses, NATs and firewalls. End-to-end communication is vital in every phase of MapReduce tasks to share input blocks or intermediate results. In this paper, we demonstrate how we guarantee end-to-end connections in WAN by using a virtual overlay network built on a P2P system [18]. In addition to the limited connectivity in WAN, resources in VC have lower bandwidths comparing to dedicated data center machines. Thus, the current node- and rack-locality based data placement and task scheduling policies of MapReduce need to be redesigned. This paper does not address this challenge; it is a subject of future work.

MapReduce Result Verification Due to the *free-to-join* nature of VC resources, task results must be confirmed correct. The majority-voting method at a central manager that is leveraged by BOINC [19] might not be applicable in MapReduce due to the large size of intermediate results. In order to deal with this issue, Tang et. al. [13] and Moca et. al. [20] proposed distributed majority voting method that can be leveraged for intermediate result verification of MapReduce on VC resources.

III. System Design

In this section, we present a heuristic to select subsets of opportunistic resources that are likely to provide reliable service by referencing the uptime of a current session. We also propose the classification of nodes into an *infant-pool* to increase the chance of locality-data usage of resources that are predicted to be less reliable.

A. Considering Uptime to Determine Volatility

Javadi et. al. [17] and Nurmi et. al. [21] analyzed the availability logs of opportunistic resources from the SETI@Home project and other computing resources. They fit the availability patterns to statistical distributions using the Maximum Likelihood Estimation (MLE), and the Weibull distribution with the *shape* parameter between 0.0 and 1.0 represents the characteristics accurately, which means the probability of a session remaining active for another time-unit increases as the longevity of the current session time increases. In other words, we can interpret the length of each available session (uptime) as the indicator of the probability that the session will remain available in the future.

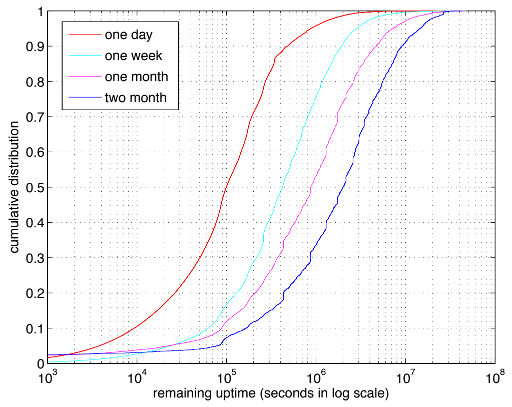

Figure 1: The CDF of remaining uptime of all session. The median remaining uptime of sessions whose running time is over one-day (the left-most line) is about 10^5 seconds. The same metric of sessions whose uptime is over two-months (the right-most line) is about 2×10^6 seconds

In order to confirm this characteristic, we analyzed the SETI@Home availability log files from *Failure Trace Archive* [5] from a perspective that was not addressed in the previous analysis by Javadi et. al. [17]. Figure 1 shows a plot of Cumulative Distribution Function (CDF) of remaining uptime given that current uptime of an available session is longer than the measured values (i.e., one day, one week, one month, and two months, from left to right). We observe that, as the current session uptime gets longer, the remaining uptime also gets longer. For instance, the median remaining time of sessions whose uptime is longer than one-day, one-week, one-month, and two-months are $10^5, 4\times10^5, 8\times10^5$, and 2×10^6 seconds, respectively. From this figure, we can observe that the uptime of an available session can be an indicator of future availability. In the evaluation section, we will further compare our proposed uptime-based resource selection method and availability rate-aware resource selection mechanism [7] with respect to the number of interruptions due to resource volatility.

We observed that longer uptime opportunistic resources in a volunteer computing project tend to stay available longer, and we can designate the long-running nodes for more important tasks at every MapReduce phase in order to decrease the interrupted service due to resource volatility. To investigate the distribution of number of resources with respect to the given uptime, we count the number of resources regarding their uptime at a daily basis in the Figure 2. The horizontal axis shows the index of days since October 5, 2008. The vertical axis shows the cumulative number of resources in log scale. The bottom bar shows the number of resources whose uptime is over two months, and the upper bars represent the number of nodes whose uptime is over one month, one week, and one day. Given

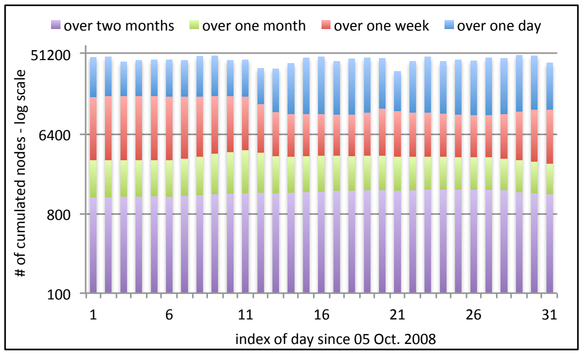

Figure 2: number of active nodes whose uptime satisfies the given criteria - 2 months, 1 month, 1 week, 1 day. At the observed 1 month of time window, over 1000 nodes have been running for over two months.

that the number of active resources at the period is about 10^6, we can see that about half of the available nodes have been running over one day. We can also see that over 1,000 nodes have been running for over two months, and over 3,000 nodes have been available over one month during the observation period. Though a direct comparison of the number of *datanodes* with typical Hadoop clusters might not make a fair illustration due to the distinct available storage size and processing capacity, it is worth pointing out that typical large-scale Hadoop clusters in production today have few thousand nodes.

In our proposed MapReduce framework on opportunistic resources, we would like to leverage the long-running nodes to increase the reliability of the system. The *namenode* and *jobtracker*, namely a central manager node, maintains a threshold value, Th_{join}, to determine if a resource has been running for long period of time, and to see if it qualifies to serve as a *datanode* and *tasktracker*. Based on the demands of a job, a central manager node can adjust the Th_{join} value dynamically; as more jobs are submitted, Th_{join} will be lowered to invite more opportunistic resources serving as *datanode* and *tasktracker*.

B. Infant Pool

Though we can increase the system reliability by using the long-running nodes as the *datanode*, the failure of a long-uptime node that was running as a *datanode* can result in resetting the uptime, and it will take a while for the node to be eligible for serving as MapReduce node. However, an interrupted node whose uptime is less than Th_{join} can still have valid data-blocks that can be exploited for data locality in task execution. According to the study of data-block popularity distribution based on age [22], a data-block is accessed by tasks for a non-negligible period of time - at least several days. In an uptime-based MapReduce resource selection mechanism, it is possible that a short-interval of interruption can make a node with valid data-blocks unable to serve as a *data-node* or *task-tracker* if the current session time is less than TH_{join}. In order to accommodate nodes with valid data-blocks but short-running times, we introduce the concept of an *infant-pool*. To compare with nodes in *infant-pool*, we name a set of nodes with uptime larger than Th_{join} as *regular pool*. Due to higher possibility of interruptions for nodes in *infant-pool*, we do not allocate new data-blocks to nodes in the pool. Instead, if task execution slots of resources in *regular-pool* are all busy and the local-block task execution condition is met for a node in the *infant-pool*, the task is performed in the infant-pool node. The data-blocks in *infant-pool* nodes are not counted towards the valid data-block so that a *namenode* tries to achieve the expected level of replication with data-blocks in a *regular-pool*.

By leveraging resources with likelihood of high-availability in the future, we expect to decrease the number of unexpected interruptions and improve system reliability. By introducing the concept of an *infant-pool*, we try to leverage already existing data-blocks for local-data task execution.

C. Implementation Details

In order to validate our proposed heuristics, we implemented a prototype using Hadoop-0.22.0 as a base source code. We added a module to record uptime of a *data-node*, and a module which determines whether a data-node is going to be located in an *infant-pool* or *regular-pool*. We leverage the default Hadoop block placement module by adding a component to distinguish nodes in infant-pool and regular-pool. The job scheduler also needs to distinguish them, and we implemented the proposed module on Hadoop's default job-queue-scheduler and delay-scheduler. At both scheduling modules, we added a variable at the *JobInProgress* class to count the number of rejected *task-tracker* scheduling attempts. Initially, the count value is 0, and the value is increased every time a job is not assigned to a given *task-tracker*. Otherwise, if a task gets assigned, the value of rejected scheduling count is decreased by half. If the number of rejected *task-tracker* is larger than the number of resources in *infant-pool* plus *regular-pool*, the job can be assigned to nodes in the *infant-pool*. By counting the number of rejected task-tracker scheduling attempts, a job can wait until a task-tracker in regular-pool tries to get assigned tasks. If task-trackers in regular-pool are busy running other jobs, the rejected count will increase, and the job can be assigned to nodes in a infant-pool. This step is similar to delay scheduler [8], which tries to increase the chance of data block local-node execution by waiting a short period of time.

IV. Evaluation

In this section, we compare the performance of node selection heuristics of our proposed uptime-based approach and availability-rate aware methods through simulation. We also demonstrate the operation of the proposed heuristics by implementing a prototype on top of Hadoop, deploying it on a real-world WAN environment, and conducting experiments by replaying a synthetic failure scenario.

A. Experiments on Uptime-based Resource Selection

In order to evaluate the performance of uptime-based resource selection mechanism, we implemented a memory-efficient simulator which can replay a SETI@Home log file by repeating the available/unavailable transitions per each node. Of the total nodes, we select a target number of resources (500, 1000, 2000, 3000) using different node selection methods. If a selected node becomes unavailable, we remove the failed node from the list and add a new high-ranked node. After performing the simulation for the entire period, we summarized the data between May 17, 2008 and October 26, 2008; this period was selected as it exhibits the largest number of active resources. The number of interruptions (when the target number of resources is 1000) is shown in Figure 3.

Each bar shows the number of interruptions for different node selection mechanisms. *rnd* means random node selection method. *up-t* is our proposed uptime-based node selection method. *upt+ar* is the combination of uptime-based method and available-rate based method. In the method, we select high uptime nodes that have been available over 95% of time in the past. *w-upt* means the weighted uptime, which reflects the past available rate while selecting nodes based on the uptime. In this method, we multiply a constant value calculated from the previous available rate to make a node that has higher previous available rate to be registered earlier. *p-ar* means the available-rate based resource selection method that excludes the current session uptime; hence, this value is static when a new session begins. *comp-ar* means comprehensive available-rate. This method considers available rate calculated by including the current session time; thus, this value is dynamic. In summary, *up-t* is our proposed node selection method. *upt+ar* and *w-upt* are uptime-based node selection methods which consider previous available rate. *p-ar* and *comp-ar* are methods that rely on the previous available-rate.

As shown in the figure, our proposed uptime-based method (*up-t*) shows the least number of interruptions comparing to other heuristics. Uptime-based methods which consider the past available rate (*upt-ar* and *w-upt*) do not show as good result as the simple uptime-based method. The available-rate aware node selection methods incurs 2.6x more interruptions than our proposed mechanism. Previous work on MapReduce on opportunistic resources, ADAPT [7], leverages available-rate to improve the reliability of the system. Based on this simulation result, we can expect that the system reliability can be improved by changing the reliable-node selection mechanism.

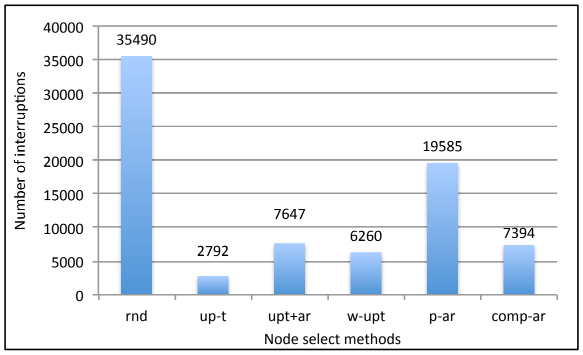

Figure 3: The total number of interruptions during 6 months with 1,000 nodes according to the different node-selection method. *up-t* is our proposed uptime-based selection mechanism. It incurs fewer than half interruptions compared to the availability-rate aware resource selection method.

In order to further investigate the reasons of different number of interruptions, we show the minimum uptime and average prior available rate among the selected nodes at the time of selection in Table I. The uptime-based methods show much higher minimum uptime than availability-rate based method. Among the uptime-based methods, the ones that consider the available-rate lower the barrier of uptime and increase the previous available-rate to be selected as MapReduce nodes. However, this lowered uptime barrier resulted in the more number of interruptions as shown in the Figure 3. We measured the Pearson product-moment correlation coefficient between the number of interruptions in Figure 3 and the minimum uptime value in Table I, and the coefficient is -0.6171, which means high negative correlations; the higher the uptime, the lower the interruptions.

Table I: The minimum uptime and average available rate of selected nodes

	minimum uptime (seconds)	average available rate
rnd	1	0.7156
up-t	4,433,460	0.76
upt+ar	184,713	0.9859
w-upt	2,425,241	0.9717
p-ar	4	0.9964
comp-ar	713724	0.999714

Figure 4 shows the average number of interruptions-per-day (primary vertical axis) with different number of target resources (horizontal axis) and the minimum uptime of the registered nodes (secondary vertical axis) for the uptime-based resource selection method. As we can see from the figure, as the target number of resources increases, the average number of interruptions increases and the minimum uptime of registered nodes decreases, which match the observations in Table I. The percentile value on top of each bar shows the fraction of nodes that have failed on a daily-basis. For

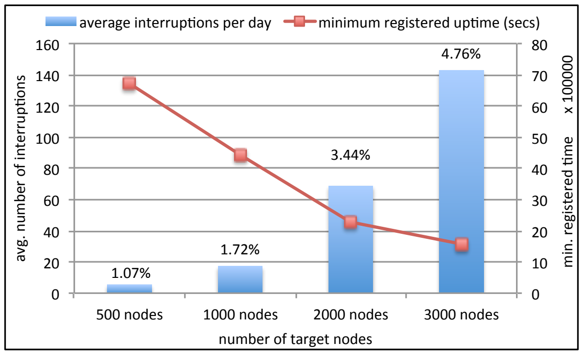

Figure 4: The average number of interruptions per day of uptime-based node selection method and the minimum uptime of registered nodes. As the target number of nodes increases, more interruptions happen and the minimum uptime decreases. With 3000 target resources, 4.67% of nodes fail daily on average.

the target number of resources equal to 500, about 1% of nodes became unavailable. For the target number of nodes being 3000, 4.76% of nodes transit to unavailable status. By carefully selecting subset of nodes with high-chances of being available in the future, we can provide reliable MapReduce services by using the opportunistic resources.

B. *Experiments with a Prototype Implementation in WAN*

In this section, we evaluate the prototype implementation described in Section III-C. In order to evaluate our proposed algorithm in a realistic WAN environment, we created a virtual cluster with 30 virtual machine instances in Chicago, Texas, and Florida (10 VMs in each site) using FutureGrid [9]. Each VM is equipped with 1GByte of memory and a single core with Hadoop-0.22.0 of our proposed algorithm implemented. The *namenode* is located on the Chicago site, and other nodes work as both *data-node* and *task-tracker*. In order to guarantee the end-to-end connection in a WAN environment, we created a virtual network using IPOP [18], and all communications happen through the virtual network. In order to verify the heterogeneous network status, we show the intra- and inter-site ping response time and average bandwidth in Table II and Table III, respectively. Note that we use *iperf* to measure the bandwidth, and these values include the overhead of processing every packet at an overlay network layer. As shown in the table, we can replay the heterogeneous latency and bandwidths in the given experiment environment.

Table II: ping response time (milli-seconds)

Ping (msec)	Florida	Chicago	Texas
Florida	0.3	46	88
Chicago	46	0.24	56
Texas	62	56	0.92

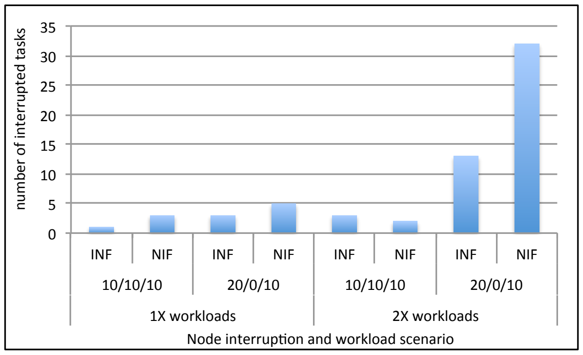

Figure 5: The number of interrupted tasks due to transition to unavailable state with different job loads and failure scenario. In general, using the *infant-pool* decreases the number of interrupted tasks.

Table III: bandwidth measurement (bits per second)

BW (bps)	Florida	Chicago	Texas
Florida	137	38.73	5.08
Chicago	17.11	152	19.63
Texas	34.39	32.84	53.37

We replayed resource availability/unavailability transitions by referencing the SETI@Home log files at the date of 14 October 2008. Because we could not replay logs of all available nodes at the time given the smaller set of 30 VM instances in the experiment, we synthetically selected 30 nodes from different previous uptime categories. In order to differentiate the prior uptime, we divide nodes in the log file into three categories; *high-uptime* nodes have over one-week of uptime before the date. *short-uptime* nodes have less than one-day of uptime at the observed date, and *mid-uptime* nodes have been running more than one-day, less than one-week at the date. At this experiment, we randomly select 10/10/10 as the number of resources at each short/mid/high uptime node category. In order to generate more transitions, we also performed experiments with node selection distribution of 20/0/10. We implemented infant-pool method both with Hadoop default job-queue scheduler and delay-scheduler. They show similar patterns, and we present the results with the default job-queue scheduler.

In order to evaluate the system in a realistic job submission scenario, we leverage SWIM [10], a statistical workload injector for MapReduce based on log files from Facebook. The workload includes various kinds of workloads that have high volume of input dataset for Map task and intermediate data for Reduce task. We generated 100 jobs from the given job scenario with different resource pool size configurations to differentiate job loads.

Figure 5 shows the number of interrupted tasks under different job loads with different node state transition rates. The Facebook job generator manipulates the job loads by

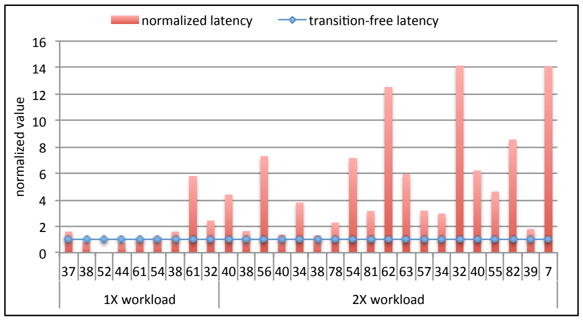

Figure 6: The impact on the job response time normalized with interrupt-free job execution when a task is interrupted due to state transition.

changing the input, intermediate, and output data size while keeping the inter-job submission rate the same. By changing the distribution of uptime nodes (10/10/10 and 20/0/10), we expect to change the rate of transition between available and unavailable status. *INF* shows the number of interrupted tasks when infant-pool is enabled, and *NIF* shows the metric when infant-pool is disabled. We can observe that the number of interrupted tasks decreases when we enable infant-pool, especially when 2X workload with more transitions happening.

In order to observe the influence of task interruption to the final job response time, we calculated the normalized job latency (Figure 6). We normalize the latency of jobs with interrupted tasks to job response time with no interruptions. Note that block placement remains the same for both cases, but different job scheduling decision can result in different job response times, especially due to heterogenous available bandwidths. As shown in the figure, jobs with interrupted tasks have a heavy impact to the response time, which motivates to minimize the number of task interruptions. In addition to the final response time impact, we can also observe that the impact varies across the jobs. This can be explained by referencing the work by Dinu et. al. [23]. They observe that *task-tracker* failures in the middle of execution affects MapReduce job response time unexpectedly based on the progress of the job when the failures happen.

Overheads: Task execution and block placement with *infant-pool* enabled can incur additional overhead with respect to the task schedule waiting time, because a task waits at least one-round of update from task-trackers until it meets *regular-pool* nodes. Given that the default *task-tracker* update interval is 3 seconds, this waiting time is negligible comparing to the MapReduce job execution time. Infant-pool also induces additional storage overhead, because a block placed in an infant-pool node is not counted as a live block. In our experiments, infant-pool enabled execution consumes at most twice more storage capacity comparing with infant-pool disabled mode.

V. Conclusion

In this paper, we presented challenges and opportunities of using MapReduce on opportunistic resources. Among the challenges, we focused on heuristics of selecting subset of highly-available nodes in the future by using the current session time. With thorough analysis of real opportunistic resource log files, we could confirm that our proposed uptime-based resource selection method can perform better than availability-rate aware prediction methods. The analysis also showed the feasibility of using opportunistic resources in MapReduce if we designate a small number of resources that are predicted to be highly available in the future. The experiments were conducted by replaying the volunteer computing availability logs with a real-world MapReduce job scenario in WAN environment with a prototype implementation of our proposed infant-pool feature with Hadoop MapReduce. The experimental results demonstrate that the infant-pool helps to decrease the number of interrupted tasks that can impact the final job response time.

Throughout the analysis and experiment, we could not only discovery the opportunities of volunteering resources for MapReduce but also some challenges that have be addressed. First, in a wide-area environment where the end-to-end bandwidth is limited, we need to re-design the task scheduling module being bandwidth aware. In a very-large volunteer computing resource pool, the current client/server architecture might have limitations for close monitoring/management of underlying resources, and we will address this issue by creating a MapReduce pool on-demand based on job demands using a decentralized resource discovery method.

Acknowledgment

This material is based upon work supported in part by the National Science Foundation under Grants No. 0910812, 0751112, 0855123 and 0855031. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[2] Apache Software Foundation. http://hadoop.apache.org/.

[3] M. Moca, G.C. Silaghi, and G. Fedak. Distributed results checking for mapreduce in volunteer computing. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, pages 1847 –1854, may 2011.

[4] Konstantin Shvachko. Apache haoop the scalability update. *USENIX, login Volume 36 No 3*, June 2011.

[5] Derrick Kondo, Bahman Javadi, Alexandru Iosup, and Dick Epema. The failure trace archive: Enabling comparative analysis of failures in diverse distributed systems. In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pages 398 –407, may 2010.

[6] Konstantin Shvachko. Hdfs scalability: The limits to growth. *USENIX, login Volume 35 No 2*, April 2010.

[7] Hui Jin, Xi Yang, Xian-He Sun, and Ioan Raicu. Adapt: Availability-aware mapreduce data placement for non-dedicated distributed computing. In *Proceedings of the the 32nd International Conference on Distributed Computing Systems*, ICDCS 2012, 2012.

[8] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *Proceedings of the 5th European conference on Computer systems*, EuroSys '10, pages 265–278, New York, NY, USA, 2010. ACM.

[9] Gregor von Laszewski, Geoffrey C. Fox, Fugang Wang, Andrew J Younge, Archit Kulshrestha, Gregory G. Pike, Warren Smith, Jens Voeckler, Renato J. Figueiredo, Jose Fortes, Kate Keahey, and Ewa Delman. Design of the futuregrid experiment management framework. In *GCE2010 at SC10*, New Orleans, 11/2011 In Press. IEEE, IEEE.

[10] Yanpei Chen, Archana Ganapathi, Rean Griffith, and Randy Katz. The case for evaluating mapreduce performance using workload suites. In *Proceedings of the 2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, MASCOTS '11, pages 390–399, Washington, DC, USA, 2011. IEEE Computer Society.

[11] Heshan Lin, Xiaosong Ma, Jeremy Archuleta, Wu-chun Feng, Mark Gardner, and Zhe Zhang. Moon: Mapreduce on opportunistic environments. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 95–106, New York, NY, USA, 2010. ACM.

[12] Simon Delamare, Gilles Fedak, Derrick Kondo, and Oleg Lodygensky. Spequlos: a qos service for bot applications using best effort distributed computing infrastructures. In *Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing*, HPDC '12, pages 173–186, New York, NY, USA, 2012. ACM.

[13] Bing Tang, M. Moca, S. Chevalier, Haiwu He, and G. Fedak. Towards mapreduce for desktop grid computing. In *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2010 International Conference on*, pages 193 –200, nov. 2010.

[14] Gilles Fedak, Haiwu He, and Franck Cappello. Bitdew: a programmable environment for large-scale data management and distribution. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, pages 45:1–45:12, Piscataway, NJ, USA, 2008. IEEE Press.

[15] Kyungyong Lee, David Wolinsky, and Renato J. Figueiredo. Pond: dynamic creation of htc pool on demand using a decentralized resource discovery system. In *Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing*, HPDC '12, pages 161–172, New York, NY, USA, 2012. ACM.

[16] D. Kondo, A. Andrzejak, and D.P. Anderson. On correlated availability in internet-distributed systems. In *Grid Computing, 2008 9th IEEE/ACM International Conference on*, pages 276 –283, 29 2008-oct. 1 2008.

[17] B. Javadi, D. Kondo, J.-M. Vincent, and D.P. Anderson. Discovering statistical models of availability in large distributed systems: An empirical study of seti@home. *Parallel and Distributed Systems, IEEE Transactions on*, 22(11):1896 –1903, nov. 2011.

[18] A. Ganguly, A. Agrawal, P.O. Boykin, and R. Figueiredo. Ip over p2p: enabling self-configuring virtual ip networks for grid computing. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, page 10 pp., 2006.

[19] David P. Anderson. Boinc: A system for public-resource computing and storage. In *Fifth IEEE/ACM International Workshop on In GRID*, 2004.

[20] M. Moca, G.C. Silaghi, and G. Fedak. Distributed results checking for mapreduce in volunteer computing. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, pages 1847 –1854, may 2011.

[21] Daniel Nurmi, John Brevik, and Rich Wolski. Modeling machine availability in enterprise and wide-area distributed computing environments. In *In Euro-Par05*, pages 432–441, 2003.

[22] Ganesh Ananthanarayanan, Sameer Agarwal, Srikanth Kandula, Albert Greenberg, Ion Stoica, Duke Harlan, and Ed Harris. Scarlett: coping with skewed content popularity in mapreduce clusters. In *Proceedings of the sixth conference on Computer systems*, EuroSys '11, pages 287–300, New York, NY, USA, 2011. ACM.

[23] Florin Dinu and T.S. Eugene Ng. Understanding the effects and implications of compute node related failures in hadoop. In *Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing*, HPDC '12, pages 187–198, New York, NY, USA, 2012. ACM.