# MARLA: MapReduce for Heterogeneous Clusters

Zacharia Fadika [1], Elif Dede [2], Jessica Hartog [3], Madhusudhan Govindaraju [4]

*Computer Science Department, Binghamton University*
*P.O. Box 6000, Binghamton, NY 13902-6000, USA*
[1] zfadika@cs.binghamton.edu
[2] ededel@binghamton.edu
[3] jhartog1@binghamton.edu
[4] mgovinda@binghamton.edu

***Abstract*—**
**MapReduce has gradually become the framework of choice for "big data". The MapReduce model allows for efficient and swift processing of large scale data with a cluster of compute nodes. However, the efficiency here comes at a price. The performance of widely used MapReduce implementations such as Hadoop suffers in heterogeneous and load-imbalanced clusters. We show the disparity in performance between homogeneous and heterogeneous clusters in this paper to be high. Subsequently, we present MARLA, a MapReduce framework capable of performing well not only in homogeneous settings, but also when the cluster exhibits heterogeneous properties. We address the problems associated with existing MapReduce implementations affecting cluster heterogeneity, and subsequently present through MARLA the components and trade-offs necessary for better MapReduce performance in heterogeneous cluster and cloud environments. We quantify the performance gains exhibited by our approach against Apache Hadoop and MARIANE in data intensive and compute intensive applications.**

## I. Introduction

Introduced in 2004 [1], MapReduce has slowly gained popularity with its swift and efficient data-intensive processing abilities. However, the performance of widely used MapReduce implementations, such as Hadoop, suffers in heterogeneous environments. This deficiency stems from the uniform application of a `map` and `reduce` function to nearly equally split data amongst participating nodes. Such data is made local to the nodes, as it is less expensive to bring the computation to the data, rather than bringing the data to the computation [2]. However, uniform `map` and `reduce` methods, being applied by all nodes holding similarly sized data, lends to performance problems when such nodes have different performance abilities. Should some nodes be faster than others, such nodes will perform and finish quicker, while the cluster waits for the slow nodes to complete their tasks before presenting the output to the user. While Hadoop [3] tries to tackle this problem with its *straggler* mitigating mechanism, [4] and [5] have shown such a mechanism to be inefficient. Similarly, in [6] we have shown Hadoop's *straggler* mitigation scheme to be inefficient in heterogeneous environments, in addition to having a limited impact in homogeneous settings turned heterogeneous with the addition of third party loads to parts of a shared cluster or cloud. This can happen with new users logging in and deploying their own applications. Other MapReduce implementations such as Twister [7] and LEMO-MR [8] altogether lack a mechanism to tackle this issue. Furthermore, solutions presented to address this problem, such as LATE [4] and [5] confront the issue with a static load-balancing approach. In [5] for instance, the cluster is assessed and its faster nodes are identified prior to application runtime. Input partition is then skewed to favor those fast nodes. Faster nodes get more data to process because they are rightly expected to perform better. Even though LATE has been shown to perform to expectations in static conditions, an obvious problem to note is the following: should previously determined fast nodes come under intense load during runtime, due perhaps to new users logging in and starting heavy jobs, the input partition cannot be adjusted, neither can the load be transferred due to very large data sizes. To address this shortcoming, we propose MARLA: (MApReduce with adaptive Load balancing for heterogeneous and Load imbalAnced clusters). In this paper, we present and discuss its architecture and design, then subsequently show that not only does MARLA outperform Hadoop in heterogeneous clusters, as it is designed to, it also outperforms Hadoop in homogeneous settings. We present our findings as we test MARLA both at the National Energy Research Computing Center (NERSC) and the Binghamton University Grid and Cloud Computing Laboratory across a wide range of experimental scenarios, against Hadoop, and MARIANE MapReduce [9], from CPU-intensive applications to data-intensive applications, to fault-prone scenarios.

The contributions of this paper are as follows:

- We present a dynamic load-balancing MapReduce implementation, suited not only for homogeneous clusters, but also for heterogeneous, and load-imbalanced environments.
- We show how the integrity and advantages of the MapReduce model can be maintained while providing good performance in heterogeneous and load-imbalanced environments.
- We present a performance comparison of our approach with existing MapReduce implementations and present the performance narrative emerging from our experiments.

## II. The Architecture of a Dynamically Adaptive MapReduce Platform

### A. MapReduce for Shared-Disk file systems

MARLA is an implementation of the MapReduce model that uses the underlying shared-file systems, such as GPFS [10] and NFS [11], as its I/O management mechanism.

Hadoop makes use of the Hadoop Distributed File System [12] as the need for data replication is essential for fault-tolerance. With Apache Hadoop, each input file chunk is replicated to the user's liking, allowing the `datanodes` to hold redundant copies of each others' input. Should a node fail, another node holding the same input chunk is called into action to preserve the integrity of the running job. The use of the HDFS also allows Hadoop to execute speculative tasks when faced with heterogeneous clusters. Chunks belonging to slow nodes are also owned by other nodes, perhaps faster nodes. Once a node is determined to be slow by Hadoop, duplicate tasks are scheduled on nodes holding the same input chunk. This, as shown in [8] [13] can be highly overhead-prone. This is so because the `Master` node needs to account for integrity tallies belonging to each block, for possibly several of them, along with their replicas, their location and the health of the nodes holding them. Nodes need to report their block condition and their own condition to the `Master`. Such operations as we discussed in [6] rob CPU cycles from the nodes, which could be directed towards processing. Furthermore, network channels used for output transfer can be obstructed by extensive communication in the case of large clusters. Under failures, blocks of input belonging to failed nodes also need to be replicated. When unavailable due to substantial node failures they need to be copied between nodes, potentially overwhelming the network when it comes to failing data-intensive applications. This analysis led us in MARLA's case, and MARIANE MapReduce [9] before MARLA, to make use of shared-file systems instead of the HDFS. The use of shared-file systems allows us to make use of more input space as replicas do not exist at the user level, in the user's directory space. For more robust fault-tolerance, more space is required. Hadoop needs dedicated node space to operate and thus can be unfriendly to already existing Grid and Cloud computing facilities not willing to rearrange their networks and disks for Hadoop's purpose. MARLA has been tested at NERSC, which supports GPFS, and also on the Grid and Cloud Computing Laboratory cluster at Binghamton University, which supports NFS.

### B. The NERSC case

Aside from logistical advantages as discussed in the previous section, the use of shared-file systems such as NFS, and GPFS also absolves the MapReduce cluster from managing the file system, thus potentially increasing its effectiveness, as all of the framework's resources are solely geared towards *mapping* and *reducing*. As previously mentioned, the use of a shared-file system with MapReduce also makes the model compatible with many Grid/Cloud computing centers, as most of them already make use of one or more distributed file system technologies espousing a node-shared disk layout. The HDFS is not a POSIX compliant file system and as such, it is not suitable for scientific and legacy applications that rely on POSIX compliant file systems in the Grid/Cloud setting. The use of HDFS is much more limited, and in a facility such as NERSC, requires resource isolation and cluster re-arrangement [9]. NERSC [14] hosts over 7 central clusters, as well as a myriad of specialized sub-clusters hosting various energy research projects. NERSC totals approximately 17,000 available nodes setup for MPI use, used for research purposes, and within which sit over 200,000 processing cores. NERSC also offers over 2000 petabytes of storage space for compute and data intensive applications. Apache Hadoop is installed on its Magellan cloud and on the Carver cluster and benefits from a subset of NERSC's computing power. This stems from Hadoop's requirement to operate under HDFS. MARLA, however, in such settings can inherently make use of all available resources with no need for revamping as its mode of operation is directly compatible with such environments. It is thus logistically more convenient and more efficient for MARLA's purposes to operate on shared-file systems. This is so, as given the same machines, both NFS in our Laboratory and GPFS at NERSC, offered better file system Read/Write performance than HDFS did in either setting. Even as we acknowledge in [9] that file system speed does not guarantee ultimate application performance when it comes to MapReduce, it can certainly help as large number of items are fetched from the disk during a data intensive application's runtime.

### C. Designing MARLA

The design of MARLA rests upon a dynamic task scheduling mechanism allowing each node to request tasks at its own pace. Contrary to the traditional MapReduce approach with Hadoop and most MapReduce implementations, where tasks are equally divided and predefined for each node before starting a given application, MARLA allows for participating nodes to request work as they complete previous tasks. As data items are derived through splitting of the input given by the user, such items become tasks. The `Master` node then registers the total number of tasks available to the nodes. Nodes are afforded a processing identification tag, and use it to request tasks. Tasks are requested for as many cores as a node possesses (an option the user can control in the application settings). As those tasks are collected by the processing nodes, they become immediately unavailable to other processing nodes, which have to move on to further tasks. Subsequently, a node does not request a task before it has successfully completed the tasks it previously requested for its cores. This scheduling scheme ensures that slow and fast nodes alike will process their fair-share. In testing, as we show in Section III, an even load distribution between homogeneous nodes, and an uneven one when slow and fast nodes, were mixed. Naturally, fast nodes did more work than slow nodes in heterogeneous cluster cases. Similarly in stressed homogeneous clusters, fast

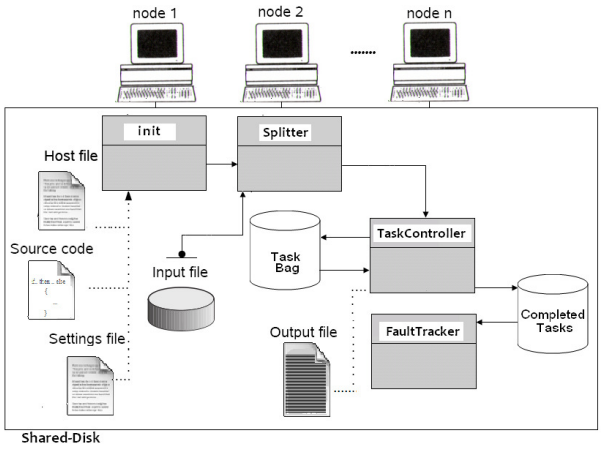


Fig. 1. Architecture used for MARLA

nodes originally start with much work, but end up requesting less work as their performance decreases due to third-party user impact. The same performance observation however, as we will show, did not occur quite efficiently for Hadoop and MARIANE. Finally, the granularity pertaining to the number of tasks per node to process for MARLA can be adjusted and depends on the user. Finer granularity means lots of smaller tasks. This is beneficial for highly heterogeneous environments and bigger input data. Coarse-grain tasks suit homogeneous settings, and small files.

### D. Architecture of MARLA

As illustrated in **Figure 1**, MARLA uses three principal modules, each one representing one of the tenets pertaining to the MapReduce model; those are:

The `Splitter` for Input/Output management, the `TaskController` for concurrency management, and the `FaultTracker` for fault-tolerance.

**Figure 1** shows the design used for MARLA

### E. Input Management

*1) Input Splitting:* While Hadoop and most MapReduce applications partition the input equally amongst nodes, then transfer each chunk to their destinations, MARLA relies on the inherent shared-file system it sits atop for this feat. MARLA, unlike Hadoop, does not produce a given number of chunks, usually similar for each node. Instead, the framework considers input chunk as tasks, and several of them, depending on the user, are created. The framework leverages the data visibility offered by the shared-disk file system to provide its input data to the cluster nodes. Input management and split distribution are thus not performed on top of an existing file system (FS), but rather left to the underlying shared file system's default mechanisms. This absolves the MapReduce implementation from the responsibility of low-level file management and from the overhead of efficiently communicating with the FS through additional system layers. Furthermore, MARLA benefits not only from file system and data transfer optimizations provided by evolving shared-disk file system technology, but can solely focus on *mapping* and *reducing*, rather than data management at a lower level.

*2) Input Distribution:* Input distribution is directly operated through the shared-file system (SFS). As the input is deposited by the user, the SFS is optimized to perform caching and pre-fetching to make the data visible to all nodes on-demand. This frees the MapReduce framework from accounting, and transferring input to the diverse nodes. Another benefit of shared-disk file systems with MapReduce, one which became apparent as the application was implemented is the following: current MapReduce implementations, because of their tightly coupled input storage model to their framework, require cluster re-configuration upon node change. As with Hadoop, nodes own their input chunks and as such fail with them. Replacing a node means replacing its input. With MARLA, nodes are simply workers, not data guardians. Should a node failure occur, a rescuing node can be brought in and will start requesting available tasks immediately. Similarly, more nodes can be instantaneously added in between runs with no need for time consuming data rearrangements if the departing nodes held large input chunks. MARLA allows for storage to be independent from the worker nodes. Separating the I/O structure from the nodes allows for a swift reconfiguration and a faster application turnaround time, when dealing with iterative MapReduce applications. In Hadoop's case, removing a node holding a crucial input chunk means finding a node holding a duplicate of the chunk held by the exiting node and copying it to the arriving node, or just re-balancing the cluster, as to redistribute the data evenly across all nodes.

### F. Task tracker and task control

The task tracker, also known as `Master`, in addition to making tasks available from data chunks provided by the `Splitter`, makes the *map* and *reduce* code written by the user available to all participating nodes through the shared-file system. This results on the application level to a one time instruction dispatch, rather than *map* and *reduce* instructions streaming to as many participating nodes as there are in the cluster. Upon launch, the nodes designated as mappers, subsequently use the *map* function, while those designated as reducers, use the *reduce* function. The task tracker monitors task progress from the cluster nodes, and resubmits failed ones into the `task bag` through the `FaulTracker` to be re-tried by other available workers. In the interim, the worker carrying the failed task is excused from processing for a short period of time. This time-out counts as a strike if any other nodes were able to successfully complete the task. Upon three strikes, the worker is "out", and no longer allowed to participate. Completed tasks are moved to the `completed task bag` and sent to the reducer which operates in a similar fashion.

### G. FaultTracker and Fault-tolerance

While Hadoop uses task and input chunk replication for the purpose of fault-tolerance, MARLA practices task specific fault-tolerance. Failed tasks are simply resubmitted into the

`task bag`, the offending nodes are put on short temporary leave while the task is re-tried. If the task succeeds with another worker, the offending node is given a strike. Three strikes and the node is considered faulty and no longer allowed to participate in the ongoing processing. This fault-tolerance scheme avoids expensive data relocation, and proves beneficial performance-wise in fault-prone scenarios. One case that became apparent to us in testing with Hadoop is the need to increase Hadoop's replication count as more nodes are expected to fail. Failing to do so would result in total application failure should a significant part of the cluster die. In our case, we recorded total failures with a 32 node cluster (128 cores total), and 5 replicas per input chunk, if such a cluster lost more than 50% of its nodes. Understandably the threshold for total failure can be raised with a higher replica count. However, a higher replica count means that much more space. MARLA's scheme does not necessitate replications and as such does not require the same space demands. Furthermore the task model we espouse in this paper means that even as we lose a significant part of our cluster, as long as one node is still alive, tasks will get requested by that node until the job is complete. Simply put, MARLA's design offers a higher total failure-threshold than Hadoop does if one cannot afford extensive file chunk replication space. A drawback however of our approach is that file-level fault-tolerance is completely dependent on the underlying shared filesystem.

## III. Distributed Large-Scale Data Processing

In this section, we test MARLA's performance in both heterogeneous and homogeneous cluster settings. We do this to show that heterogeneous cluster performance was not acquired at the expense of homogeneous cluster performance. Similarly, we test MARLA and Hadoop under failing node conditions to showcase MARLA's fault-tolerance performance against that of our past research endeavors, namely MARIANE's [9]. Given that MARLA is partly based on MARIANE, we have included it here as a comparison between not only MARLA and Hadoop, but also between MARLA and MARIANE.

As heterogeneity is most common in high performance computing due to various loads on various machines, rather than wide physical differences between nodes, we simulate cluster heterogeneity by running various third party CPU-intensive and memory intensive loads on various nodes of our testbed, using MPRIME [15], a well known benchmark and machine stress-testing tool. We subsequently test all three frameworks in a physically heterogeneous cluster and show the load management capabilities exhibited by each tested framework.

Our experiments were conducted on the National Energy Research Scientific Computing Center's cluster (NERSC), and in the Binghamton University Grid and Cloud Computing Research Lab. On NERSC, we performed our tests on the Magellan cluster where MARLA was installed on top of GPFS, and tested alongside the local Apache Hadoop v.20 installation running on the same test bed. The Binghamton University Grid and Cloud Computing Research Lab has the same Hadoop version, and hosts MARLA using NFS. We elected to consistently afflict 75% of our cluster with load, while 25% of it remained idle. Furthermore, in both facilities, exclusive access to the clusters was guaranteed to ensure the integrity of the results portrayed below. All stressed nodes received 50% CPU-utilization loads on all of their cores, while every other node saw 50% of their memory consumed by MPRIME.

In all the experiments discussed in this paper, we ran MARLA, MARIANE and Hadoop using identical nodes, identical node counts, identical input data and similar user source code.

## IV. Performance Results

We run our tests on a selection of two clusters:

NERSC Magellan cluster

- 8 core – Intel Nehalem machines, with 2.66Ghz and 24 GB of ECC RAM. The file system in use here is GPFS. Results on this class of machines are taken by averaging the timings produced on these nodes.

Grid and Cloud Computing Research Lab Cluster at Binghamton University

- Dual core – One desktop-class machine, which has a single 2.66Ghz Intel Core 6600 with 8GB of ECC RAM, and quad cores running Linux. The file system in use here is NFS v.4.
- Quad core – 1U nodes in a cluster, each of which has two 2.66Ghz Intel Xeon CPUs, 8GB of RAM, 4 cores, and run a 64-bit version of Linux. The file system in use in the test directory is NFS v.4
- 2 $\times$ quad core – 1U nodes in a cluster, each of which has two 2.6Ghz Intel Xeon CPUs, 8GB of RAM, 8 cores, and run a 64-bit version of Linux. The file system in use in the test directory is NFS v.4

Experiments 2, 3, 4, 5, 6 were run on NERSC, under GPFS with MARLA, whereas experiments 1 and 7 were run at Binghamton University under NFS.

**Figure 2**: In this heterogeneous cluster, prior to application runtime, Hadoop first partitions the same amount of work for each participating node. As the application starts running, Hadoop struggles to determine a speed standard, as nodes do not follow the same performance standard. Thrashing over load shift with speculative execution occurs, causing one of the 2 core machines involved to do more work than all of the machines on the cluster. As a consequence, the 4 core node and the other 2 core node process approximately the same load, leaving the 8 core computer, the fastest on this cluster, to process the least amount of matrices. MARLA in contrast apportions work according to the node's performance profile. The framework divides fine input chunks into tasks, and lets the participating nodes request tasks as they complete previous tasks. This results in each node processing its *fair share*, leading to the best performance. MARIANE as stated

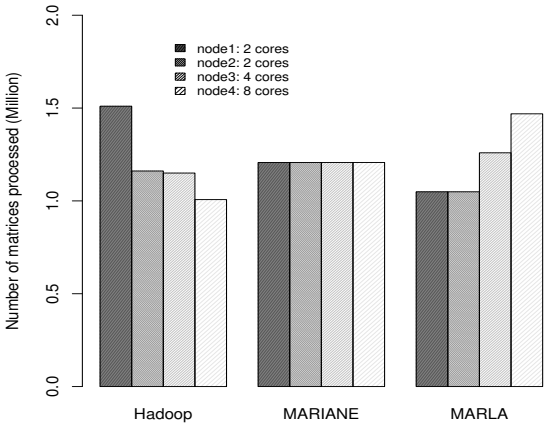


Fig. 2. 4 node heterogeneous Hadoop, MARLA and MARIANE clusters, each processing over 6 Million 33 x 33 matrices. Node1 and node2 are 2 core 2.66Ghz machines. Node3 has 4 cores at 2.66Ghz, and node4 has 8 cores at the same clock speed. All nodes feature 8GB of RAM. The load here is shown per machine. While Hadoop struggles to determine a speed standard, its slower nodes end up processing similar loads, and in the case of node1, more matrices, than its mid-range machine, the 4 core machine and its fastest machine the 8 core machine. In this case, with Hadoop, the fastest node on the cluster processes the least amount of data. MARIANE, devoid of any mechanism for heterogeneity or load balancing ends up scheduling the same amount of work regardless of node performance. MARLA on the other hand, is seen here with the two slowest nodes doing approximately the same amount of work, while the 8 core machine does most of the work, leaving the 4 core machine to sit in between the 2 core nodes and the 8 core node. The load balance here shown by MARLA is more in line with the machine's performance profile, and as such MARLA performs fastest here.

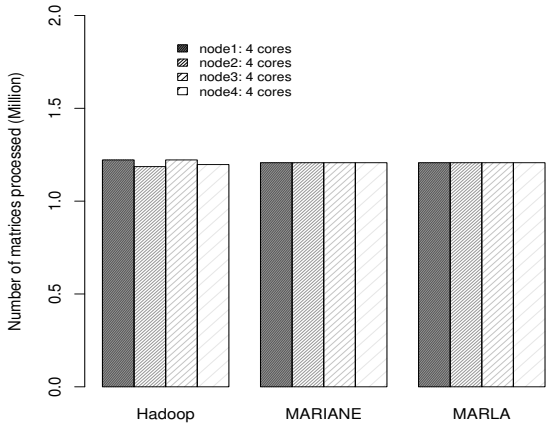


Fig. 3. 4 node homogeneous Hadoop, MARLA and MARIANE clusters, each processing over 6 Million 33 x 33 matrices. Node1, node2, node3 and node4 all 4 core machines running at the same speed - 2.66Ghz with the same memory at 8GB of RAM. The load here is shown per machine. Hadoop, MARIANE and MARLA all process about the same load per machine. Hadoop, due to a slight uneven data partition at input placement, processes slightly more data on 2 of the 4 core machines. Overall, all machines do about the same amount of work. It is worth mentioning that speculative execution occurs no matter what the condition of a job is, be it slow or fast. So this slight disparity in load processed per node could also have been speculative execution at work.

in [9] does not feature a load-balancing mechanism and thus, as Hadoop, and traditional MapReduce implementation would have it, before application runtime partitions the data equally between all participating nodes. As MARIANE lacks a load balancer, given that node3 and node4 are fastest, they have to wait for node1 and node2 to complete their tasks. MARIANE here performs worst.

**Figure 3**: In this homogeneous cluster, prior to application runtime, during input placement, Hadoop and MARIANE partition the same amount of work for each participating node. In this particular case, such a scheme is not a bad performance bearer as the cluster is homogeneous. Homogeneous nodes subjected to the same user application source code and with the same input split data sizes can be expected to process such data at a uniform speed. Although MARLA does not practice even splitting of data between nodes its *fair share* algorithm allows just about the same outcome. As homogeneous nodes progressively request tasks at their own pace, they end up processing the same amount of data.

**Figure 4** shows a 75 node (600 cores) cluster with 75% of its nodes under various CPU and memory loads, while the remaining 25% are idle prior to runtime. This experiment is meant to simulate a cloud environment where virtual machines are deployed on shared nodes, and even though homogeneous in nature, they are likely to be subjected to different loads, making them capable of widely different performance. We use

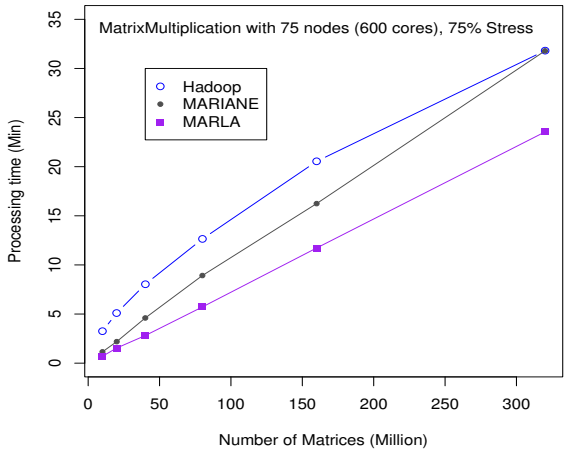


Fig. 4. 10 to 300 Million 33 x 33 matrices being gradually processed by MARIANE, Hadoop and MARLA with each 75 nodes (600 cores). This test occurs while 75% of the cluster is under different types of third-party CPU and memory loads, while the remaining 25% of the machines are idle prior to runtime. This simulate a laboratory environment where although machines can be homogeneous, due to user utilization, are most likely under different loads, and thus provide different performance. Hadoop starts upstaged by an inherently faster MARIANE, but even despite its struggles in heterogeneous environments, Hadoop catches up to MARIANE which devoid of any mechanism to balance load here, struggles even more with 320 Million matrices. While Hadoop and MARIANE take approximately 32 minutes of processing, MARLA performs best here with close to 22 minutes on the same task, with the same nodes, the same input data, and the same level of stress on 75% of its nodes.

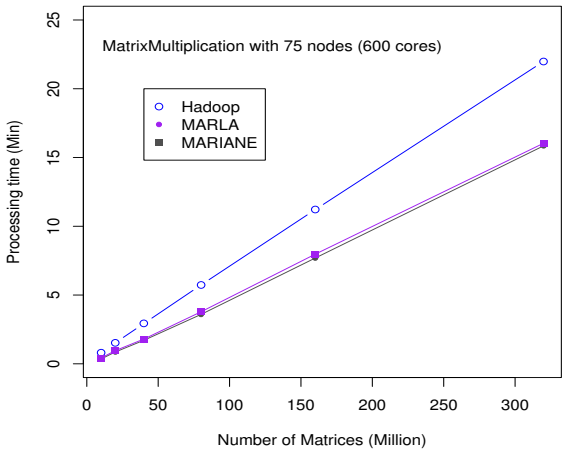


Fig. 5. 10 to 300 Million 33 x 33 matrices being gradually processed by MARIANE, Hadoop and MARLA with each 75 nodes (600 cores). This test in contrast to **Figure 4** occurs with the same 75 nodes (600 cores) under no-stress whatsoever. All machines here are idle before runtime and return to idling after runtime. As we previously showed in [9] here MARIANE's lightweight design gets the better of Hadoop. MARLA performs slightly below MARIANE as its task scheduler requires task management, in the form of responding to task requests, and recording successful tasks. MARIANE on the other hand, simply equally divides work and starts the workers. No task management takes place during runtime. MARIANE here performs best by less than a 1% margin on MARLA, while Hadoop performs about 25% worse than both MARLA and MARIANE.

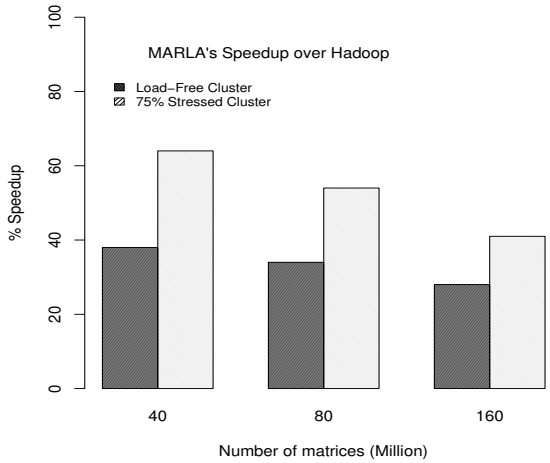


Fig. 6. Speedup between stress-free and stressed cluster runs for MARLA over Hadoop. This shows how much faster MARLA is over Hadoop in stress-free clusters vs stressed ones.

MPRIME [15] here to induce different CPU and memory loads on different machines. MARIANE even though faster than Hadoop at first due to its low overhead nature [9], performs worst in the end as Hadoop catches up to it. MARIANE is devoid of any load balancing capability and as such suffers in this scenario. Hadoop, although featuring a somewhat inefficient load-balancing model, copes better than MARIANE. Hadoop's load shifting strategy is however upstaged here by MARLA which performs best. While Hadoop's speculative execution shifts load back and forth between stressed and stress-free machines, MARLA does not operate any load shift, but lets each machine request jobs as they complete their prior tasks. This, results in the lowest runtime and thus best performance of all three frameworks.

**Figure 5** shows a homogeneous cluster of 75 nodes (600 cores), all of which are stress-free. Devoid of the need to load-balance, and due to its lightweight nature, MARIANE does best. The margin between it and MARLA here is however slightly below 1% with 300 Million matrices to process. Given that MARLA was inspired from MARIANE, this latency is the result of task scheduling. In this case, we minimized the granularity of MARLA's tasks as the environment at hand is homogeneous. In a homogeneous environment, less tasks means less scheduling. While this is ideal in a homogeneous environment, it is not in a heterogeneous cluster. The cost of managing those tasks however causes a performance footprint exhibited by MARLA against MARIANE shown here, even under MARLA's best setup for a homogeneous environment. Hadoop performs worst here; however, rather than load-balancing, as the performance narrative has been so far, instead, the lightweight nature of MARLA and MARIANE combined with the use of GPFS give both MARLA and MARIANE the upper hand. The same conclusions were found between Hadoop and MARIANE in [6].

**Figure 6** shows how much faster MARLA proves to be over Hadoop in two types of scenarios: Stress-free clusters and stressed clusters. In the stressed case, the stress is induced on 75% of the participating nodes. Although faster than Hadoop by up to 38% in the homogeneous cluster, MARLA shows here to be faster in the heterogeneous case with a little over 78% better performance over Hadoop. As mentioned in the introduction to our performance section, the stressed induced here occupies all cores of the stressed node at 50% utilization, while every other node is subjected to 25% memory utilization.

It is worth noting in **Figure 7** that as more data is present on the cluster, MARLA's file granularity is reduced. With bigger input data, previously finer input files become bigger and with bigger tasks, slow nodes do more task work, thus slowing down the job overall. We elected not to increase the file granularity setting in between data increases in this graph for consistency, as granularity increases would yield constantly finer files, which benefits MARLA in this instance. This fact is illustrated in **Figure 8**, where as tasks get finer, MARLA performs better in the stressed cluster environment it operates in. Also, as the data grows from 10 to 40 millions matrices, for the same granularity settings, the runtime disparity between adjacent granularity settings is greater for bigger input, for 2 and 4 input pieces per node. The runtime difference shows a greater disparity with 40 million matrices, than for the same granularity settings for 10 million matrices. This shows that as data grows, MARLA's granularity setting is best raised in value depending on the size of the data.

**Figure 9** shows MARLA's fault-tolerance abilities. As we

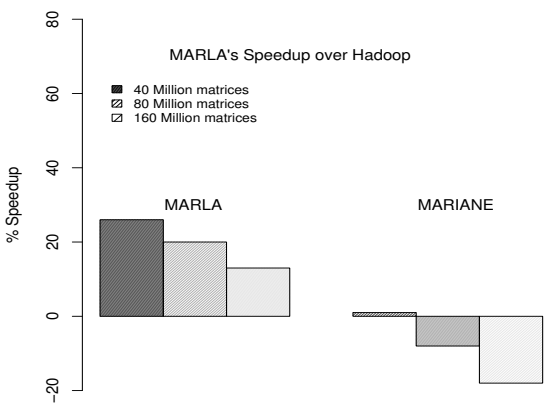


Fig. 7. MARLA's and MARIANE's performance improvements from a stress-free cluster to a stressed cluster over Hadoop. MARLA is seen here, showing a better outlook than MARIANE, as MARIANE proves to be much slower than Hadoop in stressed environments. MARLA however shows positive performance throughout the experiment. We elected to keep MARLA's file granularity settings constant in this experiment for consistency purposes. This is shown as a slight performance decrease as file sizes increase. MARLA's overall performance however shows over 16% better performance from stress-free clusters to stressed clusters over Hadoop.

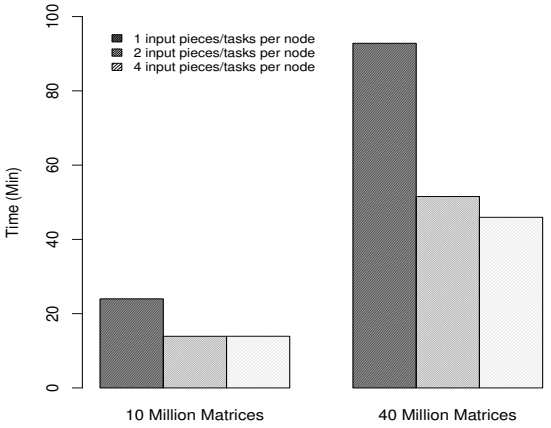


Fig. 8. The effect of task granularity on performance under load imbalanced clusters.

changed our traditional input specific fault-mechanism, as seen in MARIANE [9], to a task oriented fault-tolerance mechanism to accommodate MARLA's *fair-share* scheme, it was imperative to test this fault-tolerance scheme against Hadoop's and MARIANE's. As expected, MARIANE and MARLA perform closely, as Hadoop falls behind under failing node conditions.

## V. Related Work

While a fair body of work exists with regard to Task Scheduling in MapReduce [16] [17] [18] [19], often considering heterogeneity of clusters, these works seek to redefine task scheduling in order to improve performance when scheduling multiple jobs at once, rather than scheduling within

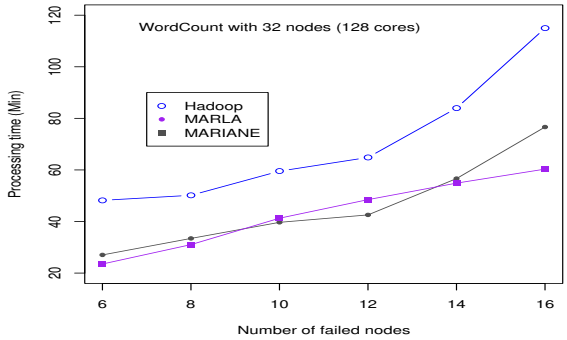


Fig. 9. Parsing of 0.6TB file full of words, in a word frequency count, under failing conditions. MARIANE Hadoop and MARLA all start with 32 nodes each, then progressively lose 6, 8, 10, 12, 14, and 16 nodes.

each individual job. Zaharia et al. [4] and Chen et al. [20] in a different approach, focus on improving speculative task execution in order for it to efficiently determine which tasks should be picked for duplicate execution. While there can be improvements to Hadoop's speculative execution model, in highly active clusters with a large number of users, nodes may idle and resume work in a constant fashion. In such settings, speculative execution would cause constant duplicate tasks to be scheduled every time nodes are detected to be burdened, even if such a condition is temporary, as the case may be. While [5] espouses a static load-balancing model by assessing the cluster and accordingly splitting the data, such an approach is inefficient should "surprise" third-party loads arise in the middle of a MapReduce job.

In the domain of MapReduce, Twister [7] is an iterative MapReduce application devoid of load-balancing as the framework focuses on catering to iterative applications. LEMO-MR [8] is a lightweight, low-overhead MapReduce framework. LEMO-MR however, much like DELMA [21], is also devoid of load-balancing. Amazon has produced EMR [22], a cloud computing framework allowing for MapReduce applications to be implemented. In a similar fashion, Microsoft has produced Azure [23]. EMR and Azure are proprietary applications, and hence an insightful analysis on their design decisions is not possible. However EMR hosts Hadoop, and as such the same approach to load-balancing as in Hadoop might be at play.

## VI. Conclusions

Used by Yahoo!, Facebook and Google, to name a few, the MapReduce model has become a widely acclaimed processing model for *big data*. As the MapReduce paradigm promotes the applications of uniform `map` and `reduce` functions to traditionally similar (size-wise) split input data chunks, it is trivial to see the source of its deficiencies in heterogeneous environments. Fortunately, the lack of performance displayed in such settings is rooted in its implementations, rather than in the model itself. In the traditional case, such as with Hadoop, Twister and MARIANE, nodes showing different performance profiles are given similar loads to process through equal data partition. Attempts at a solution to this problem

range from re-inventing Hadoop's speculative model, to static load-balancing schemes. All of these, however, mitigate the problem rather than solve it, and none directly address the performance ability of the individual worker node, which is the source of cluster heterogeneity. Faced with this condition, we presented MARLA, a load-adaptive MapReduce framework espousing a task-oriented approach to MapReduce application processing. Rather than equally splitting the input for its nodes, as in traditional MapReduce frameworks, MARLA creates a multitude of tasks born from input splits, many times greater in number than the sum of its nodes. This approach allows the participating nodes to request tasks at their own pace.

## REFERENCES


[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[2] G. Mackey, S. Sehrish, J. Lopez, J. Bent, S. Habib, and J. Wang, "Introducing mapreduce to high end computing in Petascale Data," in *Storage Workshop Held in conjunction with SC08*.

[3] Apache Hadoop. [Online]. Available: http://hadoop.apache.org

[4] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," in *OSDI*, 2008, pp. 29–42.

[5] J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian, J. Majors, A. Manzanares, and X. Qin, "Improving mapreduce performance through data placement in heterogeneous hadoop clusters," in *IPDPS Workshops*, 2010, pp. 1–9.

[6] Z. Fadika, E. Dede, M. Govindaraju, and L. Ramakrishnan, "Benchmarking mapreduce implementations for application usage scenarios," *Grid Computing, IEEE/ACM International Workshop on*, vol. 0, pp. 90–97, 2011.

[7] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox, "Twister: a runtime for iterative mapreduce," in *HPDC*, 2010, pp. 810–818.

[8] Z. Fadika and M. Govindaraju, "Lemo-mr: Low overhead and elastic mapreduce implementation optimized for memory and cpu-intensive applications," *Cloud Computing Technology and Science, IEEE International Conference on*, vol. 0, pp. 1–8, 2010.

[9] Z. Fadika, E. Dede, M. Govindaraju, and L. Ramakrishnan, "Mariane: Mapreduce implementation adapted for hpc environments," *Grid Computing, IEEE/ACM International Workshop on*, vol. 0, pp. 82–89, 2011.

[10] F. Schmuck and R. Haskin, "Gpfs: A shared-disk file system for large computing clusters," in *In Proceedings of the 2002 Conference on File and Storage Technologies (FAST*, 2002, pp. 231–244.

[11] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, "Design and implementation or the sun network filesystem," 1985.

[12] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, May 2010, pp. 1 –10.

[13] Z. Fadika, E. Dede, M. Govindaraju, and L. Ramakrishnan, "Adapting mapreduce for hpc environments," in *Proceedings of the 20th international symposium on High performance distributed computing*, ser. HPDC '11. New York, NY, USA: ACM, 2011, pp. 263–264. [Online]. Available: http://doi.acm.org/10.1145/1996130.1996166

[14] National Energy Research Scientific Computing Center. [Online]. Available: http://www.nersc.gov

[15] PrimeNet Benchmarks (GIMPS). [Online]. Available: http://www.mersenne.org/

[16] J. Polo, D. Carrera, Y. Becerra, V. Beltran, J. Torres, and E. Ayguadé, "Performance management of accelerated mapreduce workloads in heterogeneous clusters," in *39th International Conference on Parallel Processing (ICPP2010)*, 2010.

[17] J. Polo, D. Carrera, Y. Becerra, and M. Steinder, "Performance-driven task co-scheduling for mapreduce environments," in *12th IEEE/IFIP Network Operations and Management Symposium*, 2010.

[18] S. Groot and M. Kitsuregawa, "Jumbo: Beyond mapreduce for workload balancing," in *36th International Conference on Very Large Data Bases*, 2010.

[19] C. Tian, H. Zhou, Y. He, and L. Zha, "A dynamic mapreduce scheduler for heterogeneous workloads," in *Eighth International Conference on Grid and Cooperative Computing*, 2009.

[20] Q. Chen, D. Zhang, M. Guo, Q. Deng, and S. Guo, "Samr: A self-adaptive mapreduce scheduling algorithm in heterogeneous environment," in *IEEE International Conference on Computer and Information Technology*, 2010, pp. 2736–2743.

[21] Z. Fadika and M. Govindaraju, "Delma: Dynamically elastic mapreduce framework for cpu-intensive applications," in *CCGRID*, 2011, pp. 454–463.

[22] AMAZON, "Amazon Elastic MapReduce." [Online]. Available: http://aws.amazon.com/ec2

[23] Microsoft Research. [Online]. Available: http://www.microsoft.com/windowsazure/