*IOSR Journal of Computer Engineering (IOSR-JCE)*
*e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 20, Issue 4, Ver. II (Jul - Aug 2018), PP 61-68*
*www.iosrjournals.org*

# Hadoop Mapreduce for Tactical Clouds

#1 S.Swapna, ,2 **B**.Srinivas,
*Dept Of Cse, Kakatiya Institute Of Technology And Science, Warangal, Ts, India.*
*Corresponding Author: 1s.Swapna*

***Abstract***: *We envision a future where real-time computation on the battlefield provides the tactical advantage to an Army over its adversary. The ability to collect and process large amounts of data to provide actionable information to soldiers will greatly enhance their situational awareness. Our vision is based on the observation that the U.S. Military is attempting to equip soldiers with smartphones. While individual phones may not be sufficiently powerful for processing large amount of data, using the mobile devices carried by a squad or platoon of Soldiers as a single distributed computing platform, a Tactical Cloud, would enable large-scale data processing to be conducted in battlefields. In order for this vision to be realized, two issues have to be addressed. The first is the complexity of writing applications for distributed computing environments, and the second is the vulnerability of data on mobile devices. In this paper, we propose combining two existing technologies to address these issues. The first is Hadoop MapReduce, a scalable platform that provides distributed storage and computational capabilities on clusters of commodity hardware, and the second is the Mobile Distributed File System (MDFS) which allows distributed data storage with built-in reliability and security. By making the MDFS file system work with Hadoop on mobile devices, we hope to enable big data applications on tactical clouds.*

***Keywords***—*Mobile cloud, Hadoop, Map-reduce.*

---

Date of Submission: 02-08-2018 Date of acceptance: 18-08-2018

---

## I. Introduction

With advances in technology, mobile devices are becoming capable computing platforms. The new generations of mobile devices are relatively powerful with gigabytes of memory and multi-core processors. These devices have sophisticated applications and sensors capable of generating and collecting hundreds of megabytes of data. This data can range from raw application data to images, audio, video, or text files. With these enhancements in mobile device capabilities, big data processing in environments such as disaster recover sites and battlefields is becoming a reality [1]. There is currently an effort by the military to equip Soldiers with smartphones [2]. We propose utilizing these mobile devices to collect and process data in order to provide Soldiers with enhanced situational awareness. Current mobile applications that perform massive computing tasks, such as big data processing, offload data and tasks to data centers or powerful servers in the cloud [3]. Hadoop MapReduce [4] is one of the frameworks that exist to make such computation easier. It splits user jobs into smaller tasks and runs them in parallel on different nodes, reducing the overall execution time. In extreme environments, access to the traditional cloud may not be available.

Thus, the ability to carry out computation across a group of mobile devices, a Tactical Cloud carried by a squad of Soldiers or a team of first responders, is essential. This requires a Hadoop-like framework that is resilient to network failures and can operate across wireless mobile ad-hoc networks [5] typical of such scenarios. A concern that has to be addressed to enable distributed computation across mobile devices is data security, due to the envisioned applications for such systems involving sensitive information [6], [7].

Traditional security mechanisms tailored for static networks are inadequate for tactical clouds (i.e., tacticalgrade security) due to the ease with which mobile devices can be lost or captured (and data could be compromised, even if encrypted). One approach proposed to address this security vulnerability is the k-out-of-n computing framework [8] which distributes data across n nodes with the property that the data from at least k nodes is necessary to reconstruct the original information. In this paper, we replace Hadoop's native distributed file system, HDFS [9], with the Mobile Distributed File System (MDFS) [8], [10] that uses the k-out-of-n principle in order to provide the security necessary for the application domain.

In addition to the lack of tactical-grade security, a main drawback of HDFS in mobile environments is its inefficient use of resources. HDFS does not consider device energy and relies on low latency and high availability networks to replicate file blocks across multiple devices to increase reliability. Interestingly, the aforementioned k-out-of-n-enabled MDFS [8], [10] also ensures high energy efficiency. Replacing HDFS with MDFS mitigates these drawbacks while allowing Hadoop MapReduce to be used as a framework for distributed computing on mobile devices, with the following benefits: 1) parallel task execution which prevents a single device becoming a performance bottleneck; 2) efficient and fault tolerant resource management, task scheduling, and job execution; and 3) extensive testing and usage for a large number of applications over the years. The military provides a unique opportunity to leverage the power of Hadoop MapReduce operating on tactical clouds with a reliable and secure distributed file system. The opportunity arises due to the presence of a collection of mobile devices within a single domain of ownership. While it's much harder to find a group of people willing to allow their mobile phones to be used as a computing device within other domains, government issued mobile devices could be configured to be part of a distributed computing platform within the military. Such a tactical cloud would enable a number of applications to be implemented that are beneficial to Soldiers. An example of an existing application that could greatly benefit from Hadoop MapReduce in tactical clouds is the TIGR [11] system used in Iraq by deployed soldiers. This system collects information from past missions and allows for

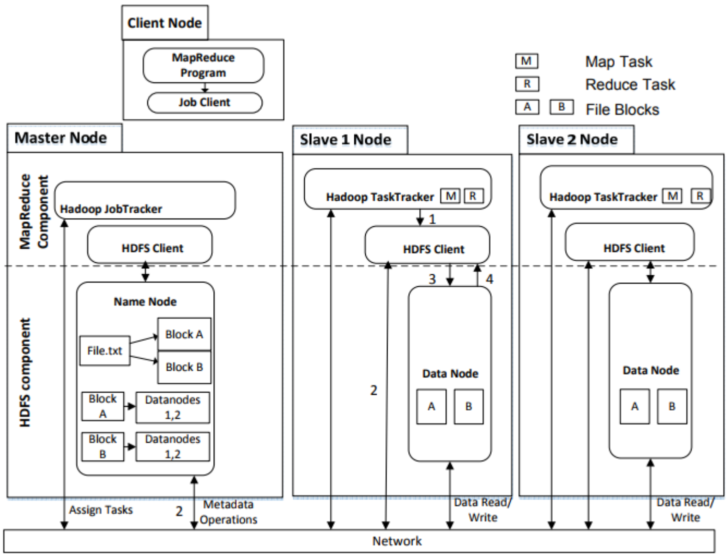



**Fig. 1. Hadoop architecture with MapReduce and HDFS components. Steps 1-4 illustrate HDFS read/write operation**

continuity of situational awareness through numerous troop rotations. Before TIGR, as troops rotate out of the theater, intelligence collected in previous missions were lost. TIGR provides a large amount of information, in the form of pictures, audio, video, and text collected over multiple missions that soldiers can manually search through. With Hadoop, the most relevant data from TIGR could be distributed across the tactical cloud using MDFS before Soldiers head out into the field. In addition, Soldiers can store new data they collect on their mobile devices. The platoon leader or squad commander could use MapReduce to extract intelligence from this data by mapping tasks such as advanced text processing or media analysis to each device, and reducing the information output by these tasks to a centralized device for visualization. In this paper, we enable Hadoop MapReduce across mobile devices by replacing its default filesystem with MDFS and evaluate its performance on a general heterogeneous cluster of devices. We modify MDFS to match the interface of HDFS, which would allow other Hadoop frameworks, such as HBase, to be used on tactical clouds. This approach also enables existing HDFS applications to be deployed across mobile devices without requiring any modifications. To the best of our knowledge, this is the first system that enables Hadoop MapReduce across mobile devices while addressing the security requirements of domains such as the military.

## II. Background, State Of Art And Challenges

### A. Hadoop and MDFS Overview

The two primary components of Apache Hadoop are MapReduce, a scalable and parallel processing framework, and HDFS, the filesystem used by MapReduce (Figure 1). Within the MapReduce framework, the JobTracker and the TaskTracker are the two most important modules. The JobTracker is the MapReduce master daemon that accepts the user jobs and splits them into multiple tasks. It then assigns these tasks to MapReduce slave nodes in the cluster called TaskTrackers. TaskTrackers are the processing nodes in the cluster that run the Map and Reduce tasks. The JobTracker is responsible for scheduling tasks on the TaskTrackers and re-executing the failed tasks. HDFS is a reliable, fault tolerant distributed file system designed to store very large datasets. Its key features include load balancing, configurable block replication strategies and recovery mechanisms for fault tolerance, and auto scalability.

In HDFS, each file is split into blocks and each block is replicated to several devices across the cluster. As shown in Figure 1, HDFS contains the NameNode and DataNode modules. The NameNode is the file system master daemon that holds the files' metadata and inode records of files and directories. An inode contains various attributes, e.g., name, size, permissions and last modified time. DataNodes are the file system slave nodes which are the storage nodes in the cluster. They store the file blocks and serve read/write requests from the client.

The NameNode maps a file to the list of its blocks and the blocks to the list of DataNodes that store them. When the HDFS client initiates the file read operation, it tries to read the block from the closest DataNodes to minimize the read latency and maximize the throughput. When the HDFS client writes data to a file, it initiates a pipelined write to a list of DataNodes chosen by the NameNode based on the pluggable block placement strategy. Each DataNode receives data from its predecessor in the pipeline and forwards it to its successor. Plain File Encrypted Encrypted AES AES Erasure Coding Secret Sharing Fig. 2. Existing MDFS architecture MDFS [12], [8], [10] is a file system that is especially suitable for battle- field computation on mobile devices provided to frontline troops. Computation occurs across a mobile ad-hoc network formed from a collection of these mobile devices, a Tactical Cloud, where each node can enter or move out of the cloud freely. MDFS is built on a k-out-of-n framework which provides energy efficiency, data security and reliability. As shown in Figure 2, every file is encrypted using a secret key and partitioned into n1 file fragments using erasure encoding (Reed Solomon algorithm). The key is also split into n2 fragments using Shamir's secret key sharing algorithm. File creation is complete when all the key and file fragments are distributed across the cluster. For file retrieval, a node has to retrieve at least k1 ($\leq$ n1) file fragments and k2 ($\leq$ n2) key fragments to reconstruct the original file. The MDFS architecture provides high security by ensuring that data cannot be decrypted unless an authorized user obtains k2 distinct key fragments. It also ensures resiliency by allowing the authorized users to reconstruct the data even after losing n1-k1 fragments of data. This scheme optimally distributes key and file fragments to the selected storage nodes such that each node contains at most one key fragment and one file fragment for each file, thereby ensuring higher reliability and security. MDFS provides a fully distributed directory service in which each node in the network periodically synchronizes its stored fragments and the corresponding key information with other nodes.

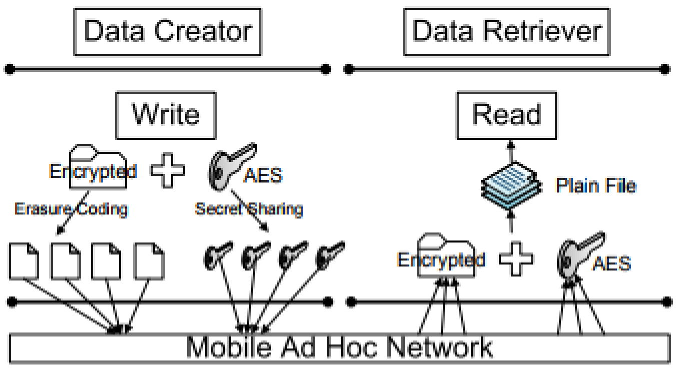



**Fig. 2. Existing MDFS architecture**

computing on smartphones. In Hyrax, Hadoop TaskTracker and DataNode processes were ported to Android smartphones while a single instance of NameNode and JobTracker were run in a single server. Such a porting of processes directly onto mobile devices does not address the shortcomings of Hadoop in mobile environments. As described earlier, HDFS is not well suited for dynamic, tactical environments. Another MapReduce framework, Misco [14] was implemented on Nokia smartphones. It has a server-client model, similar to Hyrax, where the server keeps track of various user jobs and assigns them to workers on demand. Yet another server-client model based MapReduce system was proposed over a cluster of mobile devices [15] where the mobile client implements MapReduce logic to retrieve work and obtain results from the master node. Finally, P2P-MapReduce [16] describes a prototype implementation of a MapReduce framework which uses a peer-to-peer model for parallel data processing in dynamic cloud topologies.

These solutions, however, do not solve the issues involved in the storage and processing of large datasets within the dynamic network. Huchton et al. [12] proposed a first version of a k-resilient Mobile Distributed File System (MDFS) for mobile devices targeted primarily for military operations. Chen et al. [10] proposed a new resource allocation scheme based on the k-outof-n framework and integrated it with MDFS, for significant improvements in energy consumption. We replace HDFS in Hadoop with this k-out-of-n-enabled MDFS to ensure energy efficiency, reliability, and security of Hadoop in tactical, mobile environments. For implementing the MapReduce framework over MDFS, a number of major challenges have to be addressed. The first is overcoming the limited file system functionality of MDFS, which supports only read(), write() and list(). The MapReduce framework requires a much wider range of file system operations. The MapReduce framework must also remain compatible with available HDFS applications without code modification or extra configuration. The second challenge is the fact that the MapReduce framework needs read/write streaming (i.e., reading/writing data byte by byte). MDFS can not support read/write streaming. The third challenge is to provide the JobTracker the data locality information that it needs for assigning tasks to TaskTrackers. In MDFS, since no node in the network has a complete block for processing, determining the best locations for task execution is a challenge. Finally, Hadoop uses the network topology to obtain rack awareness. If the node holding the data for processing is not available for task execution, the scheduler selects another node in the same rack. This allows the MapReduce framework to leverage the higher bandwidth of in-rack switching. Such locality is not present in MANETs due to their dynamic network topology, and thus defining rack awareness is a challenge.

## III. System Design

In the MDFS architecture, a file to be stored is encrypted and split into n fragments such that any k (fragments are sufficient to reconstruct the original file. In this architecture, parallel file processing is not possible as even a single byte of data cannot be read without retrieving the required number of fragments. Similar to the MapReduce framework which assumes that the input file is split into blocks (distributed across the cluster), we introduce blocks into MDFS. In our approach, given a configurable block size, a file is split into a corresponding number of blocks. Each block is then split into fragments that are stored across the cluster.

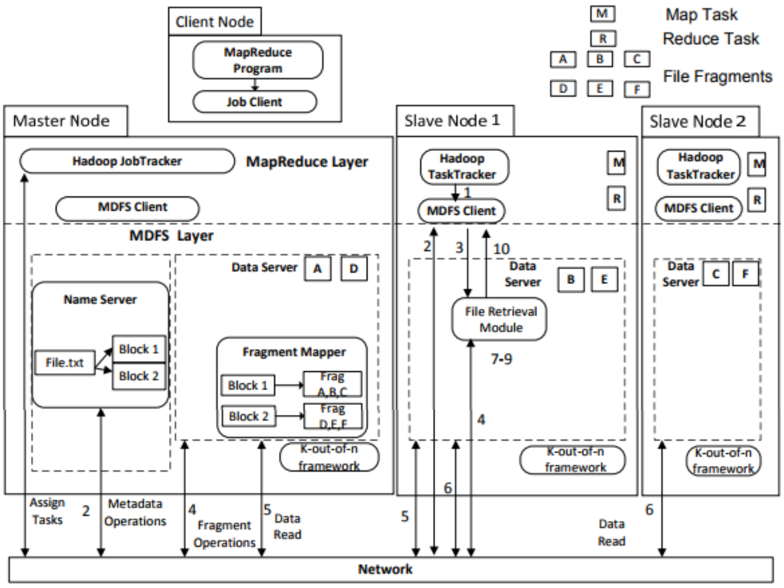



**Fig. 3. Centralized Architecture of MDFS. Steps 1-10 illustrate data read operation**

A. System Architecture We propose two approaches for our MDFS architecture: a Distributed architecture where there is no central entity to manage the cluster and a Centralized one, as in HDFS. The user chooses one architecture during the cluster startup based on the working environment. 1) Centralized Architecture: This architecture is depicted in Figure 3 which includes MDFS Client(s), a Name Server, Data Servers and a Fragment Mapper. Users invoke file system operations using the MDFS client, a built-in library that implements a file system abstraction for upper layer applications. This allows the user to be unaware of file metadata or the storage locations of file fragments. Instead, the user can reference each file by paths in the namespace. The paths use a URI format, e.g. scheme://authority/path where the scheme decides the file system to be instantiated, e.g. mdfs, and the authority is the Name Server address. The Name Server and Fragment Mapper are implemented as singleton instances across the cluster. The Name Server is a lightweight MDFS daemon that stores the hierarchical organization, or the namespace, of the file system. All file system metadata including the mapping of a file to its list of blocks is also stored in the MDFS Name Server. The Name Server has the same functionality as Hadoop's NameNode. The MDFS client and MDFS Name Server are unaware of the fragment distribution, which is handled by the Data Server. The Data Server is a lightweight MDFS daemon instantiated on each node in the cluster. It coordinates with other MDFS Data Server daemons to handle MDFS communication tasks like neighbor discovery, file creation, file retrieval and file deletion. Unlike Hadoop DataNode, the Data Server has to be instantiated on all nodes in the network where data flow operations such as reads and writes are invoked. This is because the Data Server prepares the data for these operations and they are always executed in the local file system of the client. We kept the namespace management and data management totally independent for better scalability and design simplicity.

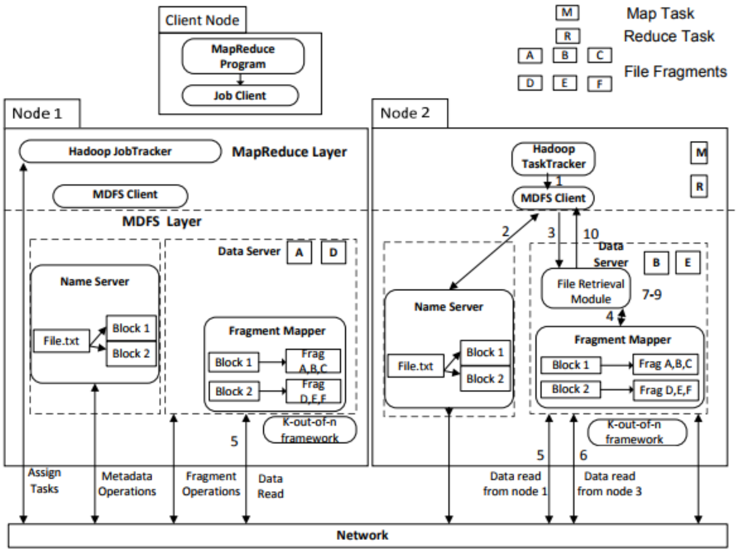
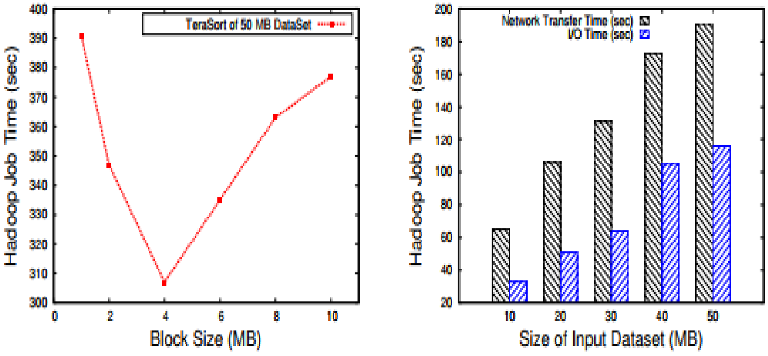



**Fig. 4. Distributed Architecture of MDFS with data read operation**

The Fragment Mapper stores information of file and key fragments which include the fragment identifiers and the location of fragments. It stores the mapping of a block to its list of key and file fragments. These daemons can be run on any node in the cluster. The node that runs these daemons is called the Master Node. MDFS stores metadata on the Master Node similar to other distributed systems like HDFS, GFS [17] and PVFS [18]. The major disadvantage of the centralized approach over the distributed approach is the master node being a single point of failure. However, this problem can be mitigated by configuring a Standby Node in the configuration file. The Standby Node is updated by the Master Node whenever there is a change in the file system metadata. The Master Node signals success to client operations only when the metadata change is reflected in both the master and standby nodes. Hence, data structures of the master and standby node are always synchronized ensuring smooth fail-over. The Master Node can become overloaded when a large number of mobile devices are involved in processing. There are several distributed systems like Ceph [19] and Lustre [20] that use multiple servers which manage the file system metadata evenly and avoid scalability bottlenecks of a single metadata server. MDFS can efficiently handle hundreds of megabytes with a single metadata server. 2) Distributed Architecture: In this architecture, depicted in Figure 4, every participating node runs a Name Server and a Fragment Mapper. The functionality (hence the description) of the MDFS Client, Name Server, Data Server, etc. is the same as in the Centralized Architecture. After every file system operation, the update is broadcast in the network so that the local caches of all nodes are synchronized. Moreover, each node periodically synchronizes with other nodes by sending broadcast messages. Any new node entering the network receives these broadcast messages and creates a local cache for further operations. This architecture has no single point of failure and no constraint is imposed on the network topology. Each node can operate independently, as each node stores a separate copy of the namespace and fragment mapping. The load is evenly distributed across the network in terms of metadata storage, in contrast to the centralized architecture. However, network bandwidth and device energy are wasted due to the messages broadcast by each node for updating the local cache of every other node in the network. As the number of nodes involved in processing increases, this problem becomes more severe, leading to higher response time for each user operation. Also, memory is wasted due to the metadata being replicated on all the devices. B. MDFS Operations 1) File Read: The design of HDFS read operation cannot be used in MDFS. For any block read operation, the required number of fragments has to be retrieved, then combined and decrypted. Unlike HDFS, an MDFS block read operation is always local to the reader as the block to be read is first reconstructed locally.

## IV. Performance Evaluation

In this section, we present performance results and identify bottlenecks in processing large input datasets. For measuring the performance of MDFS on mobile devices, we ran Hadoop benchmarks on a heterogeneous mobile wireless cluster consisting of 1 personal desktop computer (Intel Core 2 Duo 3 GHz processor, 4 GB memory), 10 netbooks (Intel Atom 1.60 GHz processor, 1 GB memory, Wi-Fi 802.11 b/g interface) and 3 HTC Evo 4G smartphones running Android 2.3 OS (Scorpion 1Ghz processor, 512 MB RAM, Wi-Fi 802.11 b/g interface). We have used Apache Hadoop stable release 1.2.1 [21] for our implementation. Our MDFS framework consists of 18,365 lines of Java code, exported as a single jar file. The MDFS code does not have any dependency on the Hadoop code base. Similar to DistributedFileSystem class of HDFS, MDFS provides MobileDistributedFS class that implements FileSystem, the abstract base class of Hadoop for backwards compatibility of all present HDFS applications. Since no changes are required in the existing code base for MDFS integration, the user can upgrade to a different Hadoop release without any conflict. We used TeraSort, a well-known benchmarking tool that is included in the Apache Hadoop distribution. Our benchmark run consists of generating a random input data set using TeraGen and then sorting the generated data using TeraSort. We considered the following metrics: 1) Job completion time of TeraSort; 2) MDFS Read/Writes Throughput; and 3) Network bandwidth overhead. We are interested in the following parameters: 1) Size of input dataset; 2) Block Size; and 3) Cluster Size. Each experiment was repeated 15 times and average values were computed. The parameters k and n are set to 3 and 10, respectively for all runs. Each node is configured to run 1 Map task and 1 Reduce task per job. As this paper is the first work that addresses the challenges in processing of large datasets in mobile environment, we do not have any solutions to compare against.




(a) (b)

**Fig. 5. (a) Effect of Block size on Job Completion Time; (b) Processing time vs Transmission time.**

A. Effect of Block Size on Job Completion Time The parameter 'dfs.block.size'in the configuration file determines the default value of block size. It can be overridden by the client during file creation if needed. Figure 5(a) shows the effect of block size on job completion time. For our test cluster setup, we found that the optimal value of block size for a 50MB dataset is 4 MB. The results show that the performance degrades when the block size is reduced or increased further. A larger block size will reduce the number of blocks and thereby limit the amount of possible parallelism in the cluster. By default, each Map task processes one block of data at a time. There has to be a sufficient number of tasks in the system such that they can be run in parallel for maximum throughput. If the block size is small, there will be more Map tasks processing less data. This would lead to more read and write requests across the network, which can be costly in a mobile environment. Figure 5(b) shows that processing time is 70% smaller than the network transmission time for the TeraSort benchmark. So, tasks have to be sufficiently long enough to compensate the overhead in task setup and data transfer for maximum throughput. For real world clusters, the optimal value of block size must be obtained experimentally.

B. Effect of Cluster Size on Job The cluster size determines the level of possible parallelization in the cluster. As the cluster size increases, more tasks can be run in parallel, thus reducing the job completion time. Figure 6 shows the effect of cluster size on job completion time. For larger files, there are several map tasks that can be operated in parallel depending on the configured block size. As shown in the figure, the increase in the cluster size results in increased performance. For smaller files, the performance is not affected much by the cluster size, as the performance gain obtained as part of parallelism is comparable to the additional cost incurred in the task setup.

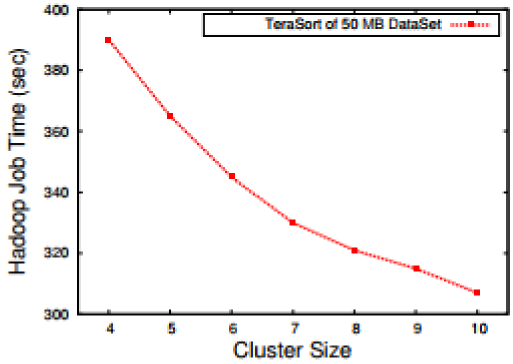



**Fig. 6. Effect of Cluster size on Job Completion Time**

C. Effects of Node Failure Rate on Job Completion Time The benchmark is run for 10 iterations for 100 MB data. Node failures are induced by turning off the wireless interface during the processing stage. This emulates real world situations wherein devices get disconnected from the network due to hardware or connection failures. In Figure 7, one, two and three simultaneous node failures are induced in iterations 3, 5 and 8 respectively and original state is restored in the succeeding iteration. The job completion time is increased by 10% for each failure but the system successfully recovered from these failures.

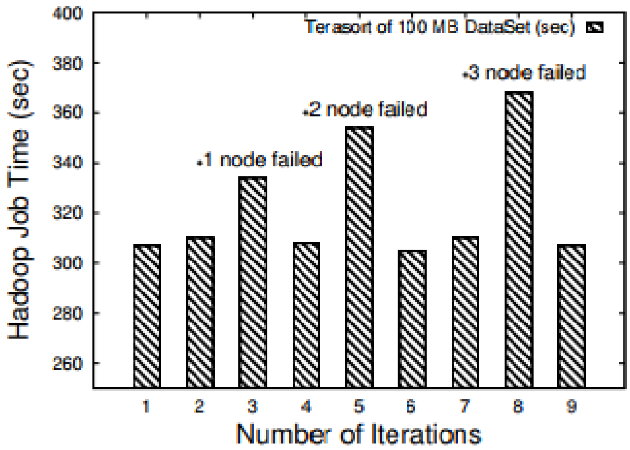
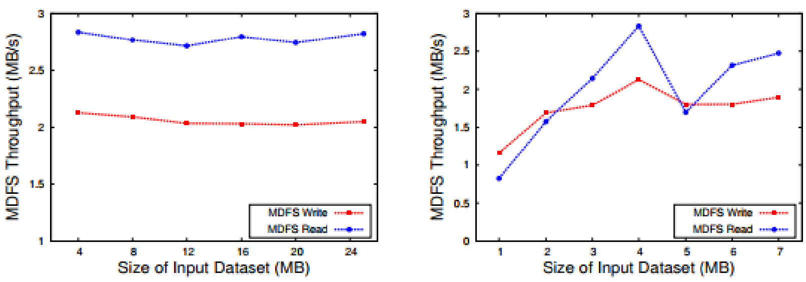



**Fig. 7. Job time vs Number of failures.**

In the MDFS layer, the k-out-of-n framework provides data reliability. If a node containing fragments is not available, the k-out-of-n framework chooses another node for the data retrieval. Since the k and n parameters are set to 3 and 10 respectively, the system can tolerate up to 7 node failures before the data becomes unavailable. If any task fails due to unexpected conditions, TaskTrackers notify the JobTracker about the task status. JobTracker is responsible for re-executing the failed tasks on some other machine. JobTracker also considers a task as failed if the assigned TaskTracker does not report the failure in configured timeout interval.




(a) (b)

**Fig. 8. MDFS Read/Write Throughput of (a) Large files (b) Small files**

## V. Conclusions And Future Work

The Hadoop MapReduce framework over MDFS demonstrates the ability of providing a Hadoop MapReduce framework in a tactical cloud where the HDFS file system is optimized to handle neither the dynamic and resource constrained nature of the tactical cloud, nor the security and reliability requirements of the domain. The evaluation results show that our system is capable of enabling big data analytics of unstructured data like media files, text and sensor data in tactical environments.

## References

[1]. G. Huerta-Canepa and D. Lee, "A virtual cloud computing provider for mobile devices," in Proc. of MobiSys, 2010.
[2]. B. McGarry, "Army set to introduce smartphones into combat," http://www.military.com/, March 2013. [Online]. Available: http://www.military.com/daily-news/2013/03/27/army-set-tointroduce-smartphones-into-combat.html
[3]. K. Kumar and Y.-H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" Computer, 2010.
[4]. "Apache hadoop," http://hadoop.apache.org/.
[5]. S. George, W. Zhou, H. Chenji, M. Won, Y. O. Lee, A. Pazarloglou, R. Stoleru, and P. Barooah, "DistressNet: a wireless ad hoc and sensor network architecture for situation management in disaster response," Communications Magazine, IEEE, 2010.
[6]. J.-P. Hubaux, L. Buttyan, and S. Capkun, "The quest for security in ´ mobile ad hoc networks," in Proc. of MobiHoc, 2001.
[7]. H. Yang, H. Luo, F. Ye, S. Lu, and L. Zhang, "Security in mobile ad hoc networks: challenges and solutions," Wireless Communications, IEEE, 2004.
[8]. C. A. Chen, M. Won, R. Stoleru, and G. Xie, "Resource allocation for energy efficient k-out-of-n system in mobile ad hoc networks," in Proc. ICCCN, 2013.
[9]. K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in Proc. of MSST, 2010.
[10]. C. A. Chen, M. Won, R. Stoleru, and G. G. Xie, "Energy-efficient faulttolerant data storage and processing in dynamic networks," in Proc. of MobiHoc, 2013.
[11]. B. Ewy, M. Swink, S. Pennington, J. Evans, J. Kim, C. Ling, S. Earp, and M. Maeda, "TIGR in Iraq and Afghanistan: Network-adaptive distribution of media rich tactical data," in Proc. of MILCOM, 2009.
[12]. S. Huchton, G. Xie, and R. Beverly, "Building and evaluating a kresilient mobile distributed file system resistant to device compromise," in Proc. MILCOM, 2011.
[13]. E. E. Marinelli, "Hyrax: Cloud computing on mobile devices using mapreduce," Master's thesis, School of Computer Science Carnegie Mellon University, 2009.
[14]. T. Kakantousis, I. Boutsis, V. Kalogeraki, D. Gunopulos, G. Gasparis, and A. Dou, "Misco: A system for data analysis applications on networks of smartphones using mapreduce," in Proc. Mobile Data Management, 2012.
[15]. P. Elespuru, S. Shakya, and S. Mishra, "MapReduce system over heterogeneous mobile devices," in Software Technologies for Embedded and Ubiquitous Systems, 2009.
[16]. F. Marozzo, D. Talia, and P. Trunfio, "P2P-MapReduce: Parallel data processing in dynamic cloud environments," J. Comput. Syst. Sci., 2012.
[17]. S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," SIGOPS Oper. Syst. Rev., 2003.
[18]. P. H. Carns, W. B. Ligon, III, R. B. Ross, and R. Thakur, "PVFS: A parallel file system for linux clusters," in Proc. of Annual Linux Showcase & Conference, 2000.
[19]. S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in Proc. of OSDI, 2006.
[20]. "Lustre file system," http://www.lustre.org.
[21]. "Hadoop 1.2.1 release," http://hadoop.apache.org/docs/r1.2.1/release notes.html.

S.Swapna, " Hadoop Mapreduce for Tactical Clouds. "IOSR Journal of Computer Engineering (IOSR-JCE) 20.4 (2018): 61-68.