# Implementation of Optimized Mapreduce With Smart Speculative Strategy And Network Levitated Merge

**Mr. Ruturaj N. Pujari, Prof. S. R. Hiray**

*Student, Pune University, Sinhgad College of Engineering, Vadgaon (BK), Pune, 411041.Maharashtra, India*
*Ruturajpujari999@gmail.com*
*Professor, Pune University, Sinhgad College of Engineering, Vadgaon (BK), Pune, 411041.Maharashtra, India*
*srhiray.scoe@sinhgad.edu*

---------------------------------------------------------------------***---------------------------------------------------------------------

***Abstract—*** ***two performance parameters of MapReduce are job execution time and cluster throughput. Straggler machine impacts these parameters. Speculative execution overcomes this by backing up those slow tasks on alternative machines. The Paper introduces new strategy Maximum Cost Performance (MCP).MCP uses i) The progress rate and the process bandwidth within a phase to select slow tasks, ii) Exponentially weighted moving average (EWMA) for prediction of process speed and to calculate a tasks remaining time, iii) A cost-benefit model to determine which task to backup based on the load of a cluster using. To choose proper worker nodes for backup tasks, it considers both data locality and data skew. The second technique is Network levitated merge. This introduces merge without merge data without repetition and disk access. Hadoop-An acceleration framework that optimizes Hadoop with the plug-in component for fast data movement. This Hadoop will double the throughput.***

**Keywords—*****Hadoop, MapReduce, straggler, speculative strategy, cluster throughput, cost performance, network-levitated merge.***

## 1. INTRODUCTION

### 1.1 Background

MapReduce is a widely used parallel computing framework for large-scale data processing. The two major performance factors in this are job execution time and cluster throughput. Straggler machines are responsible for affecting these— machines on which execute slow. Speculative execution is a common way for dealing with the straggler problem which simply back-ups those slowly executing tasks on other nodes. Multiple solutions were proposed, but they have some lacking's i) To identify slow tasks it uses average progress rate while in reality, the progress rate can be unstable and misleading, ii) If there exists data skew among the tasks, it cannot handle the situation properly iii) while choosing backup worker nodes it doesn't consider finishing time of backup task.

To address straggler's issue, add to another methodology, maximum cost performance (MCP), which enhances the effectiveness of speculative execution. In MCP: i) consider progress rate and process bandwidth to identify slow tasks, ii) Use exponentially weighted moving average (EWMA) to foresee process speed and figure out task's remaining time, iii) Determine which task to backup in given on the cluster using cost-benefit model.

To address the issue of serialization Hadoop-A is there, an acceleration framework. A novel network-levitated merge algorithm is been introduced. Also, a full pipeline is designed to overlap the shuffle, merge, and reduce phases.

### 1.2 Motivations

#### 1.1.1 stragglers problem

Google proposed MapReduce [3] in 2004,it's a popular parallel computing framework for large-scale data processing. In a normal MapReduce, the master node divides the input into various map tasks, and then schedules map tasks and reduce tasks to worker nodes in a cluster to achieve parallel processing.

Stragglers are the machines that complete the tasks in longer duration that a normal node can complete. This degrades the performance of Hadoop regarding job execution time and cluster throughput. Speculative execution strategy handles this problem.

Microsoft Dryad is another parallel computing system which supports MapReduce. Its unique speculative execution procedure is like that in Google MapReduce [3]. Later, Mantri [6] proposes another speculative execution system for Dryad. The primary distinction in the middle of

LATE and Mantri is that Mantri utilizes process bandwidth the task's (processed data /time) to compute remaining time the task's (data left/process bandwidth).

### 1.2.1 A Serialization in Hadoop Data Processing

Hadoop endeavors to pipeline the data processing. It is, in reality able to do as such, especially for map and shuffle/merge stages. As shown in Fig., after a brief initialization period. For the rightness of the MapReduce programming model, it is important to guarantee that the reduce stage does not begin until the map stage completes all data splits. In any case, the pipeline, as shown in Fig., contains a certain serialization.

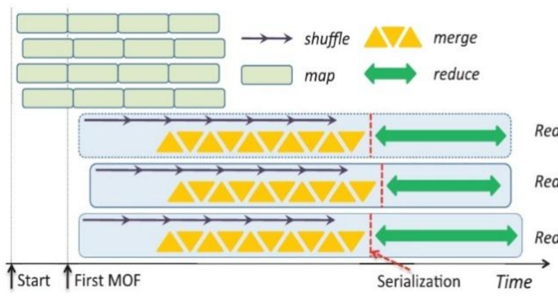


Fig.1 Serialization between shuffle/merge and reduce phases.

### 1.2.2 Repetitive Merges and Disk Access

Hadoop Reduce Tasks blend data segments when the number of segments or their aggregate size goes over a limit. However, Hadoop's current merge algorithm leads to repetitive merge, along with this comes additional disk access. Fig. Demonstrates a typical succession of merge operations in Hadoop.

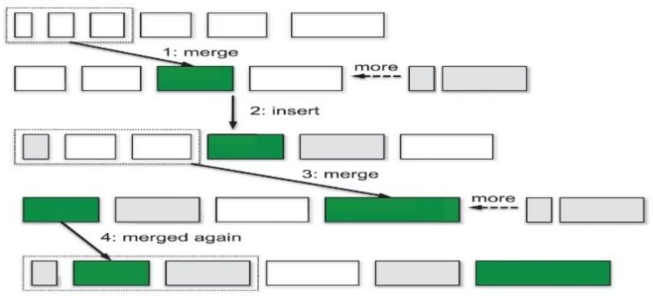


Fig 2. Repetitive merges.

### 1.2.3 The Lack of Network Portability

Hadoop only supports TCP/IP protocol;it does not support other transport protocols such as RDMA on InfiniBand [7] and 10-Gigabit Ethernet (10GigE). Hadoop-A has a C implementation which supports these networks.

## 2. LITERATURE SURVEY

Several speculative execution strategies were proposed in the literature, including MapReduce in Google [3], Hadoop [4], LATE [3], Dryad in Microsoft [3] and Mantri [6].

The 1st speculative execution strategy used in Hadoop-0.20. It just identifies straggler when tasks progress is less than average progress. But some previous studies found that this is not worth in heterogeneous environment and proposed Hadoop-LATE [3]. This stores progress rate (process/time) of the task and calculates the remaining time and similarly selects the slow tasks

Above study shows different studies done on Hadoop optimization and the limitations or lacking's of those studies. Also gives the short explanation about techniques analyzed in these papers.

These articles help to understand the working of Hadoop in different environments like, if data at local nodes how it behaves, also how it reacts to heterogeneous data.

In this paper, we made changes in working in Map phase with the help of Maximum Cost Performance (MCP) mechanism. And in reduce phase with Network-Levitated Merge (NLM) algorithm. And to achieve better performance of NLM we implemented the Hadoop acceleration.

## 3. PROPOSED MODEL

Proposed model optimizes performance in two stages Hadoop as

1. Map Phase.
2. Reduce Phase.

### 3.1 Map Phase:-

This phase uses the speculative strategy proposed in this project. The proposed method uses the MCP technique for identifying the straggler machine. The working of this module is as follows.

#### 3.1.1 To Select Backup Candidates

In MCP, we predict process speed of the task in the near future instead of using the previous average rate. In MCP, EWMA (exponentially weighted moving average) scheme is used which is expressed as follows:

$$Z(t) = \propto * Y(t) + (1-\propto) * Z(t-1)\ , 0 < \propto \leq 1 \ ......................(1)$$

Where Z (t) is the estimated and Y (t) is the observed process speed at time t, respectively. ∝ Reflects a tradeoff between stability and responsiveness.

To show effect of EWMA to predict the process speed and the remaining time of the task, system runs a demo job like sort job in cluster and suddenly gives IO and CPU intensive jobs to worker node for better prediction

### 3.1.2 To Identify Slow Tasks Using Per-Phase Process Speed

Use of progress rate and process bandwidth together minimizes the limitations of each other.

This solution, compares process speed of a task (both the processing bandwidth and the progress rate) and estimate its remaining time. Therefore, we use EWMA scheme to predict the process speed of each phase in a task, and to indentify the slow task we use this per-phase process. Meanwhile, we usethe process speed and the remaining data to process in a task to compute the remaining time of each phase in a task, and sum up the remaining time of all these phases to get total remaining time of the task.

### 3.1.3 to Estimate Task Remaining Time and Backup Time

In MCP, a task that has the longest remaining time can get the highest priority to be backed up. As watched, a task's remaining time is evaluated by the whole of the remaining time left of the every stage. When a task is running in some phasecp (i.e., the current phase), the remaining time left in cp is evaluated by the components of remain data and the processing bandwidth in cp. However, the remaining time of the accompanying stages is hard to compute as the task has not entered yet. Hence, we use the phase average process speed to calculate the remaining time of a phase (est_time$_p$). The average process speed of the phase is the average process speed of task that has entered the phase. For that phase that no task has processed, no need to compute their remaining time, which is reasonable to all tasks. Since task might handle distinctive measure of data, adjusting$\frac{est_temp_P}{\mathrm{factor_d}}$, this ration is of the input size of this task to the average input size of all tasks. Now remaining time of the tasks is calculated as follows:

$$remTime = remTime_{cp} + remTime_{fp}$$

$$= \frac{\mathrm{remdata_{cp}}}{\mathrm{bandwidth_{cp}}} + \sum_{\mathrm{p\ in\ fp}} \mathrm{est\ time_p} * \mathrm{factor_d} \text{ ........................ } (2)$$

$$factor_d = \frac{data_{input}}{data_{avg}} \text{..................................................... } (3)$$

To estimate the backup time of a slow task, use the sum of est_time$_p$ for each phase in this task as estimation. Therefore, calculate the backup time as follows:

$$backupTime = \sum_p estTime_p * factor_d \text{ ............................ } (4)$$

As seen, process speed in reduce phase decreases as time goes on. So process speed fluctuation will occur and impact the precision of time estimate. To avoid such impact, compute the remaining time of copy phase. Calculate the remaining time of the copy phase using the following equation:

$$remTime_{copy} \frac{finishPersent_{map} - finishPercent_{copy}}{processSpeed_{copy}} \text{ .................. } (5)$$

In the above equation, the process speedcopy is estimated by EWMA.

### 3.1.4 Maximizing Cost Performance of Cluster Computing Resources

Design a cost-benefit model that decides backing up of tasks and saves the status of both original and backup tasks i.e., two slots were used remTime and BackupTime. Therefore, define the profit of the two actions as follows:

$$profit_{backup} = \propto * (rem\_Time - Backup\_Time) - \beta * 2 * backupTime \text{ ....................................................... } (6)$$

$$profit_{non\_Backup} = \propto * 0 - \beta * rem\_Time \text{ ............................ } (7)$$

In above equation, rem_time and backup time are already known and ∝ and $\beta$ are the weight of benefit and cost, respectively.

Then choose the action that gains more profit. And choose proper backup node.

$$profit_{backup} > profit_{not\_backup} \leftrightarrow \frac{rem\_Time}{backup\_Time} > \frac{\alpha+2\beta}{\alpha+\beta} \text{ ........ } (8)$$

Where, $1 \leq \left(\frac{\alpha+2\beta}{\alpha+\beta}\right) \leq 2$.To simplify this formula, we replace $\frac{\beta}{\alpha}$by $\gamma$. Then the formula is$\frac{remTime}{backupTime} > \frac{1+2\gamma}{1+\gamma}$. fWhen a cluster is idle and has free slots, the cost or speculative execution is not considered, because it doesn'taffect other task's performance. On the other hand, when the cluster is busy and has many pending tasks of other jobs, the cost is an important issue because backing up a task will take more time to do the job. We assume that g varies with the load of the cluster: $\frac{1+2\gamma}{1+\gamma}$gets its lowest value $1(\gamma = 0)$ when the load of the cluster is low while reaches its highest value .we set $\gamma$ to the load_factor of the Hadoop cluster:

$$\gamma = LoadFactor = \frac{number_{pending_{task}}}{number_{free_{slots}}} \ldots\ldots\ldots\ldots\ldots\ldots(9)$$

Where,

$number_{pending\_tasks}$= the number of pending tasks,

$number_{free\_slots}$ = the number of free slots in the cluster.

When the cluster is idle and has many free slots, $\gamma$ decreases to 0 , so the backup condition becomes rem_time>backup_time. When the cluster is busy and has many pending tasks of other jobs, $\gamma$ increases and $\frac{1+2\gamma}{1+\gamma}$ converges to 2 gradually. Then the backup condition will berem_time> 2 * backup_time. As a result, less tasks will be backed up. Hence, using load_factor as g works perfectly.

After going through all the running tasks, we get the set of backup candidates. The candidate with longest remaining time are backed up finally.

### 3.1.6 To Select Worker Nodes for Proper Backup

To extract the better performance, we have to assign fast worker nodes to do backup tasks. To achieve this, use the moving average process bandwidth of Data-local map tasks completed on a worker node to represent the node's performance.

Also, consider the data-locality of map tasks when making the backup decisions. In MCP, while assigning the tasks to backup nodes, we estimate the time that this node will take to complete the task. We will give the backup task to the node if and only if it is estimated to finish faster.

## 3.2 Reduce Phase:-

During this stage system does the tasks like shuffling, merging. Here it lessens the time head required for doing these activities. For this reason, the system utilizes the Network levitated merge algorithm. The working of this algorithm is been characterized as follows. Likewise to handle the parallel processing inside of the system this proposition utilizes the pipelining system, i.e. this pipelines the shuffle, merge and reducing procedures.

### 3.2.1 Network-Levitated Merge [NLM]

This algorithm avoids the repetitive disk access while in processing. Also stays at local disks while processing the data.

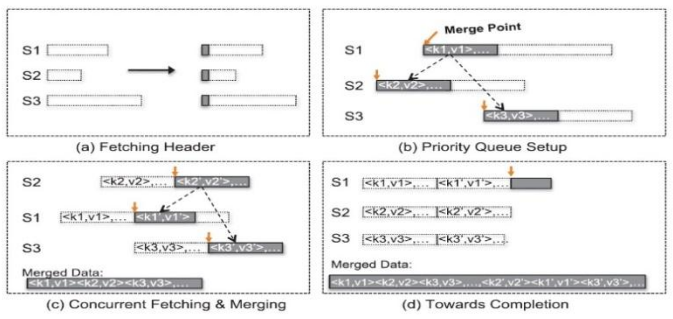


Fig 3.Network levitated merge

In Fig. 3a, three remote segments S1, S2, and S3 are to be fetched and merged. Instead taking them to local disk NLM only takes header containing partition length, offset, and the first pair of <key,val>.These <key,val> pairs are sufficient to construct a priority queue (PQ) to organize these segments. This information is sufficient to form a priority queue (PQ) to organize these segments. This algorithm does not have to store or merge segments onto local disks. Instead of merging segments when the number of segments is over a threshold, we keep building up the PQ until all headers arrive and are integrated. As soon as the PQ has been set up, the merge phase starts. The leading <key,val> pair will be the beginning point of merge operations for individual segments, i.e., the merge point. This is shown in Fig. 3b. According to <key, value>pair the segments are merged as shown in fig. 3c. When merged the data will be seen as shown in fig. 3d.

### 3.2.2 Pipelined Shuffle, Merge, and Reduce

Instead of avoiding repetitive merges, this algorithm takes off serialization barrier occurring in merge and reduce. As in Section 3.1, the merged data have <key, val> pairs ordered in their final order and can be delivered to the Java-side ReduceTask as soon as they are available. Thus, the reduce phase no longer has to wait until the end of the merge phase.

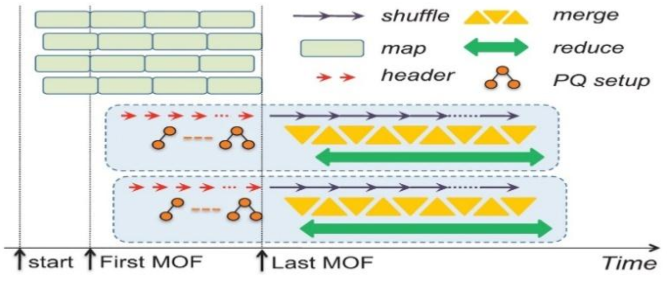


Fig4 pipelined shuffle, merge and reduce

In view of the possibility to closely couple the shuffle, merge, and reduce phases, they can form a full pipeline as shown in Fig. 4. In this pipeline, MapTasks map data split as soon as they can. When the first MOF is available, ReduceTasks fetch the headers and build up the PQ. These activities are pipelined. Header fetching and PQ setup are pipelined and overlapped with the map function, but they are very lightweight, compared to shuffle and merge operations. As soon as the last MOF is available, completed PQs are constructed. The full pipeline of shuffle, merge, and reduce then starts. One may notice that there is still a serialization between the availability of the last MOF and the beginning of this pipeline. This is inevitable in order for Hadoop to conform to the correctness of the MapReduce programming model. Simply stated, before all <key,val> pairs are available, it is erroneous to send any <key,val> pair to the reduce function (for final results) because its relative order with future <key,val> pairs is yet to be decided.

Therefore, our pipeline is able to shuffle, merge, and reduce data records as soon as all MOFs are available. This eliminates the previous serialization barrier in Hadoop and allows intermediate results to be reduced as soon as possible for final results.

### 3.3 Software Architecture of Hadoop-A

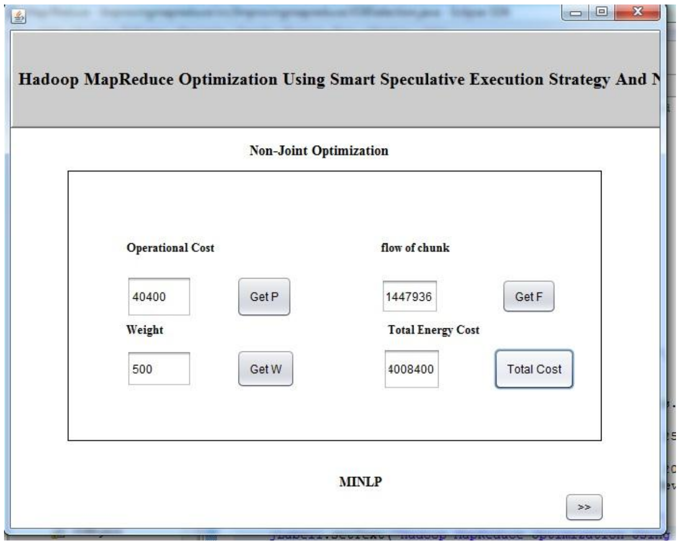
As fig.6 depicts architecture contains two components as MOFSupplier and NetMerger are threaded C implementations. This is to gain the support of RDMA. These two are developed as native C programs that are launched by TaskTracker. Use of this can be controlled by the user with help of parameter im the config. file. We can also run Hadoop without hadooop-A.

Hadoop-A supports RDMA platforms, for Infiniband, TCP/IP interconnects. Infiniband supports zero-copy data transfer. RDMA gives us the advantage to access the remote processes' memory buffers.

While the other left half works as normal Hadoop architecture, which involves the tasks like shuffling, merging of data segments.

Hadoop-A transport layer has a server in the MOFSupplier and client in the NetMerger, at client side one thread works for fetching/connection establishing request from remote server.at server side, one thread is dedicated to listen incoming request.

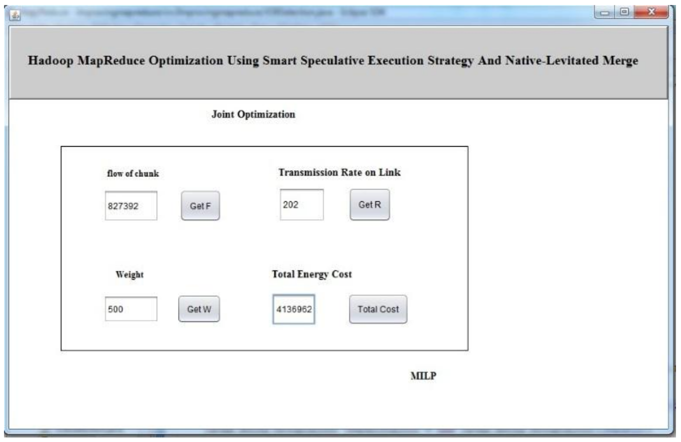
## 4. Result and Discussion


Hadoop MapReduce Optimization Using Smart Speculative Execution Strategy And N

Non-Joint Optimization

Operational Cost: 40400
flow of chunk: 1447936
Weight: 500
Total Energy Cost: 4008400

FIG. Without Joint of two method


Hadoop MapReduce Optimization Using Smart Speculative Execution Strategy And Native-Levitated Merge

Joint Optimization

flow of chunk: 827392
Transmission Rate on Link: 202
Weight: 500
Total Energy Cost: 4136962

**FIG. with joint of two methods**

From both the images we can see that the results before the methods are combined and after they are combined

The graph for the results are shown in the blue line which is slighet straight which means the map and reduce tasks are in linear forms and the red line shows the combined method results of method .

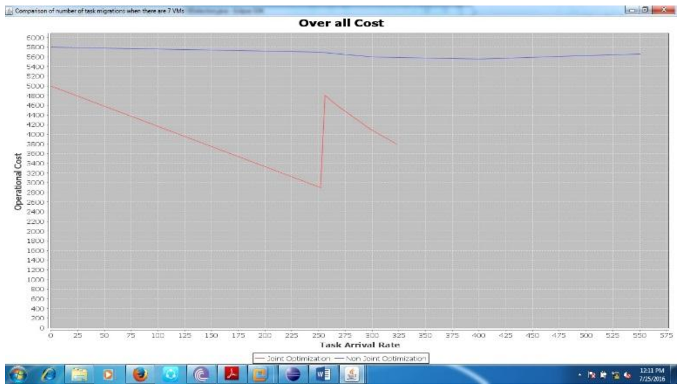


**Fig . results**

## CONCLUSION

The proposed model designed to provide the better performance while handling the straggler problem. By using MCP method and to minimize the overhead between map and reduce phases with help of network levitated merge algorithm and pipelining technique. This MCP uses EWMA for calculating the speed of worker nodes to identify the proper straggler. Network levitated merge, merges the partitioned data by just fetching the headers of each block of data. The system also uses the pipelining of shuffle, merge, and reduce phases which helps to parallel execution of shuffle merge and reduce phases to improve the performance. Proposed system is portable to any network protocol for this it uses the Hadoop-A implementation which is c based.

## REFERENCES

1. **Qi Chen, Cheng Liu, And Zhen Xiao**, *"Improving MapReduce Performance Using Smart Speculative Execution Strategy ",* IEEE Transactions On Computers, Vol. 63, No. 4, April 2014 .

2. **Weikuan Yu, Ieee, Yandong Wang, And XinyuQue**, "*Design And Evaluation Of Network-Levitated Merge For HadoopAcceleration ",* . IEEE Transactions On Parallel And Distributed Systems, Vol. 26, No. 3, March 2014

3. **J. Dean And S. Ghemawat,***"MapReduce: Simplified Data Processing On Large Clusters"*, Proc. Sixth Symp. Operating System Design And Implementation (Osdi 04), Pp. 137-160, Dec. 2004

4. **Apache**HadoopProject,*"Http://Hadoop.Apache.Org/ ",* 2013.

5. **M. Isard, M. Budiu, Y. Yu, A. Birrell, And D. Fetterly,** "*Dryad: Distributed Data-Parallel Programs From Sequential Building Blocks* ", Proc. Second AcmSigops/Eurosys European Conf. Computer Systems (Eurosys 07), 2007

6. **G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, And E. Harris,** "*Reining In The Outliers In Map-Reduce Clusters Using Mantri ",* Proc. Ninth Usenix Conf. Operating Systems Design And Implementation, (Osdi 10), 2010.

7. **Infiniband** Trade Association, "*Http://Www.Infinibandta.Org*. ", 2013

8. **M. Zaharia, A. Konwinski, A.D. Joseph, R. Katz, And I. Stoica,** "*Improving MapReduce Performance In Heterogeneous Environments* ", Proc. Eighth Usenix Conf. Operating Systems Design And Implementation, (Osdi 08), 2008.

9. **T. Condie, N. Conway, P. Alvaro, J.M. Hellerstein, K. Elmeleegy, and R. Sears,** *"MapReduce Online",* Proc. Seventh USENIX Symp. Networked Systems Design and Implementation (NSDI), pp. 312-328, Apr. 2010.29

10. **R. Recio, P. Culley, D. Garcia, and J. Hilland***"An RDMA Protocol Specification (Version 1.0) "*, Oct. 2002.

11. **B. Nicolae, D. Moise, G. Antoniu, L. Bouge, and M. Dorier,***"Blobseer: Bringing High Throughput under Heavy Concurrency to Hadoop Map-Reduce Applications ",* , Proc. IEEE Intl Symp. Parallel Distributed Processing (IPDPS), Apr. 2010.

12. **J.-A. Quiane-Ruiz, C. Pinkel, J. Schad, and J. Dittrich,***"Rafting MapReduce: Fast Recovery on the Raft "*, IEEE 27th Intl Conf. Data Eng. (ICDE), Apr. 2011.

13. **. Ranger, R. Raghuraman, A. Penmetsa, G.R. Bradski, and C. Kozyrakis,***"Evaluating MapReduce for Multi-Core and Multiprocessor Systems ",*Proc.IEEE 13th Intl Symp. High Performance Computer Architecture (HPCA 07), pp. 13-24, 2007.30