Improving MapReduce Performance in Heterogeneous Network Environments and Resource Utilization

Zhenhua Guo, Geoffrey Fox
School of Informatics and Computing
Indiana University Bloomington
Bloomington, IN USA
{zhguo, gcf}@cs.indiana.edu

Abstract— **In MapReduce, map and reduce tasks are assigned to map and reduce slots hosted by worker nodes. Usually the numbers of map and reduce slots are carefully chosen to gain optimal resource usage. We found resource utilization is inefficient when there are not enough tasks to fill all task slots as the resources "reserved" for idle slots are just wasted. We propose resource stealing which enables running tasks to steal the unutilized resources and return them when new tasks are assigned. It exploits the opportunistic use of the otherwise wasted resources to improve overall resource utilization and reduce job execution time. Besides, our practical use of Hadoop shows the current mechanism adopted to trigger speculative execution creates many unnecessary speculative tasks that are killed soon after creation as the original tasks complete earlier. To alleviate the issue, we propose Benefit Aware Speculative Execution which predicts the benefit of launching new speculative tasks and greatly eliminates unnecessary runs of speculative tasks. Finally, MapReduce is mainly optimized for homogeneous environments and its inefficiency in heterogeneous network environments has been observed in our experiments. We investigate network heterogeneity aware scheduling of both map and reduce tasks. Overall, our goal is to enhance Hadoop to cope with significant system heterogeneity and improve resource utilization.**

Keywords- heterogeneity, MapReduce, Hadoop, resource utilization, scheduling

I. INTRODUCTION

To support data-intensive applications, several frameworks, such as Google File System (GFS) /MapReduce [1, 2], Hadoop [3] and Cosmos/Dryad [4] have been proposed. Their native support of data locality aware scheduling dramatically reduces data movement. They have been used in large scale to run both commercial applications [2, 5] and scientific applications [6, 7]. Hadoop is a popular open-source implementation of GFS and MapReduce. Hadoop Distributed File System (HDFS), which is modeled after GFS, is a distributed file system designed and optimized for write-once-read-many accesses of large files that are partitioned into equally-sized chunks. MapReduce is designed for data parallel applications that can be expressed with primitive *map* and *reduce* operations. Each map task applies user-implemented map operation to the data contained in one chunk.

Modern servers are usually equipped with multi-core processors, so multiple computation tasks can run on a single node concurrently without incurring severe resource contention. However, the concurrency needs to be carefully tuned to avoid resource underuse and overloading and achieve optimal utilization. The optimal setting depends on not only hardware configuration but also application workload. Rather than guess the optimal setting automatically, Hadoop gives users the freedom to control concurrency. Each worker node has a number of map and reduce slots which can be configured by users on a per-node basis. Nodes with heterogeneous hardware can be configured individually. At any time, each slot can run one task and each task can only be scheduled to one slot. This mechanism cannot fully explore the processing capability of nodes when all slots are not used, because the resources corresponding to idle slots are just wasted. To improve resource utilization, we propose *resource stealing* which dynamically expands and shrinks the usable resource set of each running task.

Hadoop tackles fault-tolerance by speculatively launching duplicate tasks for the tasks deemed to be stragglers. Whenever a task completes, all other duplicate tasks are immediately terminated. Therefore job execution can continue in the face of task failure. The progress rates of tasks are used to identify outliers in Hadoop, which has been found by us to be insufficient because some slowly progressing tasks that are close to completion are speculated unnecessarily. We propose Benefit Aware Speculative Execution which only launches speculative tasks if they are expected to complete earlier than the speculated tasks.

Heterogeneity is inevitable for organizations that own various generations of IT resources purchased over different time frames. Another source of heterogeneity is cloud bursting, which allows organizations to dynamically scale out their infrastructures by integrating their own dedicated resources and rented resources from third-party providers (e.g. Amazon, Rackspace). Hadoop does not perform well in heterogeneous environments [5, 8]. In our study, we propose network heterogeneity aware scheduling algorithms to improve the performance of Hadoop in heterogeneous network environments.

II. RELATED WORK

Work stealing enables idle processors to steal computational work from other processors, which yields better load balancing and higher resource utilization [9]. MapReduce adopts a different model and supports finer control of tasks running on each node. We take a "reverse" approach in the sense that running tasks actively steal

available resources (e.g. CPU cores, memory, network) reserved for prospective tasks. Cycle stealing harnesses the available workstations by supplying them with work and receiving produced results [10]. Task splitting, which dynamically adjusts the granularity of map tasks based on real-time slot availability, yields better load balancing across nodes [11]. While our proposed resource stealing shares similar motivations, it is applied to the resources located on the same node and thus complementary to task splitting.

Speculative execution has been applied to various areas. Branch predictors predict which branch a conditional jump will go to and speculatively execute the instructions before the evaluation of branch condition completes [12]. For a critical task which many other tasks depend on directly or transitively, task duplication, which redundantly executes the task at multiple places, has been proposed to reduce communication cost [13]. In MapReduce, speculative execution is used for both fault tolerance and performance improvement [2]. To improve MapReduce in heterogeneous environments, Longest Approximate Time to End (LATE) chooses the tasks that hurt the response time most to speculate and schedules them to fast nodes [5]. In addition, LATE limits the number of speculative tasks. Our proposed Benefit Aware Speculative Execution goes one step further by reducing the number of useless speculative tasks.

Prefetching and pre-shuffling are proposed in [14] to increase the data locality in Hadoop. Intra-block prefetching prefetches data that will be processed subsequently by running map tasks; while intra-block prefetching prefetches whole data chunks to nodes where map tasks are expected to run. The increase of the overlap between computation and data IO yields shorter job execution time. Pre-shuffling reduces the amount of shuffled data by scheduling map tasks close to future reduce tasks which intermediate data will be shuffled to. Although heterogeneity is not mentioned in the paper, prefetching can potentially be used to "smooth out" the variation in network bandwidth. Network throughput information is explicitly used in our study to mitigate the influence of diverse network connections. For cloud environments where heterogeneity is frequently observed, Purlieus optimizes the allocation of virtual machines in a locality-aware manner to reduce the data transfer time [15].

III. Our Approaches

A. MapReduce and Hadoop Background

Hadoop adopts a master-worker architecture. *Job Tracker* that runs on the master node manages all slave nodes and jobs. A *Task Tracker* runs on each worker node and periodically reports node and task status to the job tracker. When a task tracker reports that it has available map slots, the scheduler needs to decide which queued task will be scheduled there. The default policy follows the principle of "move computation to data" and thus favors data locality.

B. Resource Stealing

As we introduced in section I, in Hadoop each worker node comprises a user-configurable number of map and reduce slots to which tasks are assigned. You can imagine that a portion of underlying resources are reserved for each slot. Whenever a task is scheduled and starts to run, the corresponding share of resources is used. Resource usage is optimal when all slots are used. By default, each node has two map slots and two reduce slots. We found the default settings are too conservative and make Hadoop not able to fully utilize the resources in our clusters. So they were changed to fall between 1x and 2x the number of cores, which is a best practice recommended by Hadoop developers.

Even if map and reduce slots are configured optimally, the resource usage of the whole system is optimal only when all slots are used simultaneously. Usually clusters are neither idle nor fully loaded. According to the collected usage data of a thousand-node Google production cluster, CPU utilization is between 0% and 50% most of the time [16]. Applying this fact to Hadoop, we can conclude that a significant portion of slots are left idle during off-peak hours.

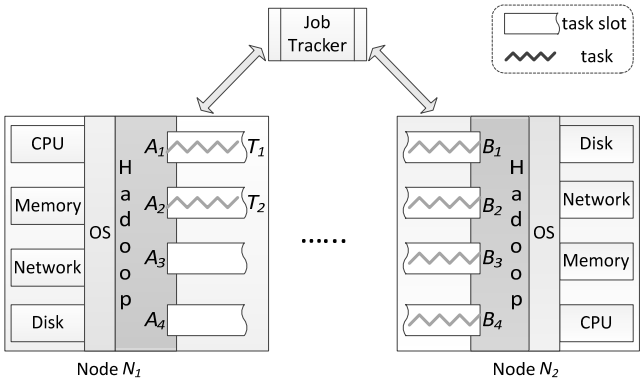
To utilize the idle resources to accelerate the execution of jobs, we propose *resource stealing* illustrated in Fig. 1. It shows how resource stealing can improve resource utilization compared to native Hadoop. In Fig. 1(a), node N_1 has four task slots $\{A_1, A_2, A_3, A_4\}$ among which two slots $\{A_1, A_2\}$ are occupied with running tasks and the other two sit idle. Vanilla Hadoop leaves idle slots unutilized unless there are new tasks to fill them. Fig. 1(b) shows how resource stealing can alleviate the issue. For each of tasks T_1 and T_2, a new task is created dynamically so that the total number of tasks becomes equal to the number of slots. Those newly created tasks can utilize the resources "reserved" for idle slots A_3 and A_4. Please note that slots A_3 and A_4 are still idle from the perspective of the job tracker. Each original task and its newly created tasks process the input data concurrently. In Fig. 1(c), two new tasks T_3 and T_4 are scheduled to node N_1 by the job tracker. To avoid severe resource contention, two running tasks are terminated, thereby handing back the stolen resources to T_3 and T_4.

We call the portion of unutilized resources *residual resources*. The next issue we investigate is how to allocate residual resources to running tasks in the system. The allocation policies can range from simple to complex in their collection and use of system state information. The policies we came up with are summarized in Table I.

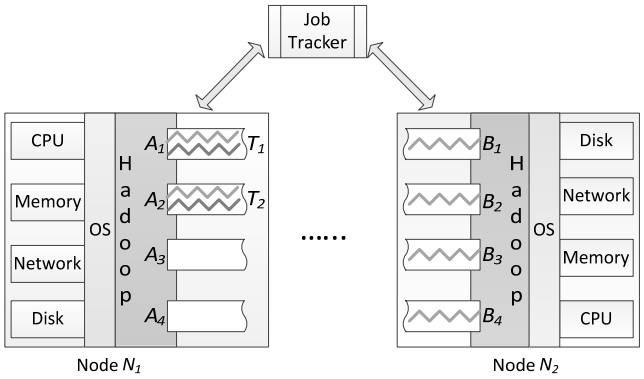
TABLE I. Resource Allocation Policies

Strategy	Description
Even	Evenly allocate residual resource to tasks
First-Come-Most	The task that starts earliest is given residual resource.
Shortest-Time-Left-Most	The task that will complete soonest is given residual resource.
Longest-Time-Left-Most	The task that will complete latest is given residual resource.
Speculative-Task-Most	Speculative tasks are given residual resource.

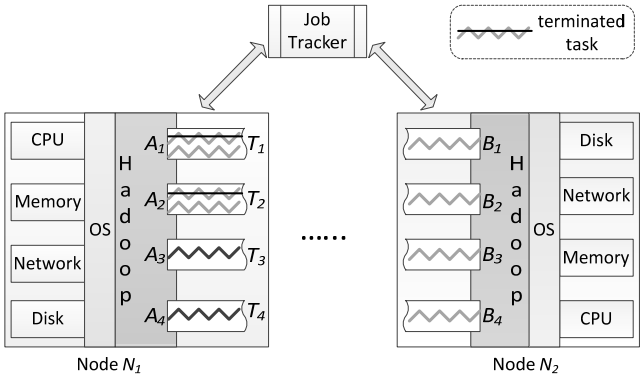
Even: This policy evenly allocates residual resources to running tasks. It is inherently stable because of not relying on the collection or prediction of system state (and thus not impacted by the information inaccuracy).



(a) Native Hadoop scheduling



(b) Steal resources when there are idle slots



(c) Hand back resources when new tasks are assigned

Figure 1. Native Hadoop vs. Resource stealing

First-Come-Most (FCM): The tasks with earliest start time are given residual resources. This policy tries to enforce the order in which jobs are submitted to the system (in a FIFO manner).

Shortest-Time-Left-Most (STLM): The task with the earliest complete time is given residual resources. The motivation is to make way for long tasks by letting tasks that are near completion complete as soon as possible. In addition, it increases the processing throughput.

Longest-Time-Left-Most (LTLM): The task that is expected to complete latest is given residual resources.

Policies STLM and LTLM require the prediction of the remaining execution time of running tasks. The approach used in [5] is adopted shown in (1) and (2).

The policy Even is simple in the sense that it does not require system state information. The fact that this policy is extremely simple does not necessarily indicate it will perform significantly worse than other more sophisticated policies. To evaluate the effectiveness of these policies in real clusters is part of our future work.

$$progress\_rate = progress / elapsed\_time \quad (1)$$

$$time\_to\_end = (1 - progress) / progress\_rate \quad (2)$$

C. Benefit Aware Speculative Execution

Speculative execution is adopted by Hadoop as a fault-tolerance mechanism. The job tracker keeps track of the progresses of all running tasks. Whenever a task is found to be unusually slow compared with peer tasks, a *speculative task* is created and launched to process the same input data. The tasks whose progress rates are one standard deviation lower than the mean of all tasks are deemed as straggler tasks. So Hadoop does not evaluate which of the speculated and speculative tasks will complete first. Consider the scenario where there are two running tasks T_1 and T_2. T_1 progresses slow with rate 1 but its progress is 90%; T_2 progresses fast with rate 5 and its progress is 50%. With default policy, Hadoop will launch a speculative task T_1' for T_1. We assume T_1' and T_2 progress equally fast. By doing simple calculation, we can find that T_1 completes earlier than T_1' although T_1 progresses much slower. The reason is T_1 is close to completion when it is speculated.

We propose Benefit Aware Speculative Execution (BASE) to resolve the issue. The remaining execution time of running tasks is calculated using (1) and (2). The expected execution time of speculative tasks is estimated using historical information. Given a regular task T_i, a speculative task T_i' of T_i, and a node N_j, FT_{ij} denotes the set of completed tasks that belong to the same job as T_i and ran on N_j. If FT_{ij} is non-empty, the harmonic mean of the execution time of tasks in FT_{ij} is used as the estimated execution time of T_i'. If set FT_{ij} is empty, the harmonic mean of all tasks of the job is used. Basically, we use the historical information of completed tasks to predict how long a speculative task will run on a specific node. We chose to adopt harmonic mean because the average of task progress rates is desired. Now we compare the expected completion time of T_i and T_i', and T_i' is executed only when it will completes earlier than T_i. With moderately accurate prediction of execution time, we expect BASE to substantially reduce the number of useless speculative tasks.

Part of our future work is to investigate the effectiveness of BASE by conducting intensive experiments on real clusters.

D. Heterogeneity Aware Scheduling

As we discussed before, clusters are not always homogeneous, and there are situations where heterogeneity

is inevitable. In hierarchical MapReduce [17], multiple local clusters, which are administrated under different domains and may even be geographically scattered, are unified to form a single MapReduce cluster that offers more processing capability than any of the participating cluster. One natural consequence is the heterogeneity of network – inter-cluster bandwidth is usually dramatically different from intra-cluster bandwidth.

We took another approach to build a unified Hadoop cluster. ViNe [18], which supports bi-directional communication between any pair of nodes, was used to create a virtual network environment across multiple physical clusters. From the perspective of applications, ViNe is transparent and thus indistinguishable from physical networks in functionalities. However, bandwidth and latency among nodes still depend upon physical network topology. In our experiments, several FutureGrid clusters were connected using ViNe. Inter-cluster bandwidth was set to 1Mbps – 10Mbps while intra-cluster bandwidth was 100Mbps – 1Gbps. Apparently the network environment was drastically heterogeneous. We deployed Hadoop on top of ViNe without modification and conducted a series of tests to evaluate performance. The scenario where data locality is 100% was chosen as the reference case. Random scheduler was developed with the support of configurable *randomness*. Randomness 0.5 and 1 mean half of tasks and all tasks are randomly scheduled respectively. We ran an IO intensive application *linecount* which counts the number of lines contained in input data. The computation in *linecount* is trivial and data IO dominates the overall execution. We varied the number of map slots per node and measured the slowdown of random scheduling. Results are shown in Fig. 2. The performance degrades 2000 – 3000 times. The reason is that some tasks greatly lagged behind other data-local tasks as they lost data locality and needed to fetch input data from nodes located in another cluster.

Although we chose application *linecount* in our tests, the experiment results are applied to not only *linecount* but also other IO intensive applications for which data IO dominates and the inefficiency in data fetching can substantially degrade the overall performance.

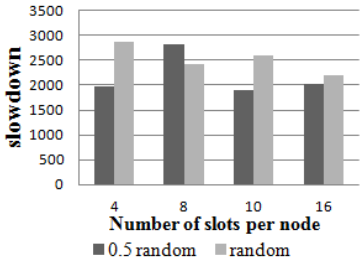


Figure 2. Inefficiency in the face of heterogeneous network

To alleviate the performance degradation, we propose network heterogeneity aware scheduling algorithms. Firstly, available network bandwidth between any pair of hosts is maintained and future network performance is estimated. Network Weather Service [19] fits this purpose which can collect real-time network bandwidth information without injecting an overwhelming number of probing packets into network and forecast future performance. Secondly, we define some useful symbols. If task T_i is assigned to node N_j, DS_i denotes the size of input data of task T_i; DT_{ij} denotes the data transfer time; PT_{ij} denotes the computation time; ET_{ij} denotes the execution time; AT_j denotes the time when node N_j will become available; and CT_{ij} denotes the wall-clock completion time. The available bandwidth between a pair of nodes N_i and N_j is denoted by $BW_{i,j}$. Their relationship is shown in (3), (4) and (5).

$$DT_{ij} = DS_i / BW_{data-src,j} \tag{3}$$

$$ET_{ij} = PT_{ij} + DT_{ij} \tag{4}$$

$$CT_{ij} = ET_{ij} + AT_j \tag{5}$$

1) Scheduling of Map Tasks

If there are available slots and none of waiting tasks can achieve data locality, we get into the dilemma of scheduling non-local tasks immediately or delaying the scheduling with the hope to achieve better data locality in the subsequent scheduling. Delay scheduling, which postpones the scheduling of non-local tasks by a small amount of time, improves data locality greatly in a system where most of jobs are small [20]. However, that approach has a couple of issues. Firstly, for systems with different workload (e.g. many large jobs), delay scheduling may not work well. Secondly, the length of delay interval is specified by users, and the optimal setting is difficult to find for heterogeneous environments and may even change with the change of system state (e.g. peak hours vs. off-peak hours).

We propose a more adaptive approach which reacts to the change of real-time system state. Given task T_i and node N_j, the expected completion time of T_i on N_j can be calculated using (3), (4) and (5). Assume there are n nodes totally. The node that yields the earliest expected completion time for a task is called the *preferred node*, formulated in (6). The set of tasks whose preferred nodes are N_j is called the *candidate task set* of N_j and denoted by C_j. If C_j contains multiple tasks, the scheduler needs to choose one of them. To find the optimal solution is NP-hard [21]. We propose two heuristics to find approximately good solutions, inspired by previous research on the scheduling of bag-of-tasks [21, 22]. The first heuristic is to choose the task with minimum expected completion time among C_j, shown in (7). This heuristic changes the availability status by the least amount and is expected to yield high throughput measured by the number of tasks processed per time unit on average. However, long tasks may get starved and priority boosting can be used to avoid starvation. The second heuristic is to choose the task with maximum expected completion time among C_j, shown in (8). This heuristic schedules long running tasks early so that they will not slow down the overall job execution. Neither of the heuristics is expected to perform consistently better than the other. We will evaluate their relative merits for different applications in the future.

$$PN_i = \arg\min_j CT_{ij} \ (1 \le j \le n) \quad (6)$$

$$\text{Heuristic 1: } \arg\min_i CT_{C_i j} \ (1 \le i \le |C_j|) \quad (7)$$

$$\text{Heuristic 2: } \arg\max_i CT_{C_i j} \ (1 \le i \le |C_j|) \quad (8)$$

2) Shuffling and Scheduling of Reduce Tasks

Each map tasks generates intermediate key/value pairs which are shuffled to reduce tasks. The only constraint of the shuffling process is that key/value pairs with identical keys must be sent to and processed by the same reduce task. This is needed to guarantee the correct semantics of MapReduce model. For each job, the distribution of intermediate key/value pairs on map side among all nodes is formulated as a matrix shown in Fig. 3. Set $\{K_1, K_2, ..., K_n\}$ contains all unique keys generated by map tasks. Set $\{N_1, N_2, ..., N_m\}$ contains all nodes in the system. Among the data generated by map tasks on node N_i, V_{ij} is the number of key/value pairs whose keys are K_j. If no map task runs on node N_i or map tasks running on N_i do not generate key/value pairs with key being K_j, V_{ij} is 0. To find the optimal way to schedule shuffling, we need to (i) choose the nodes where reduce tasks will run; (ii) figure out how to distribute intermediate data to reduce tasks. Ibrahim et al. proposed a heuristic algorithm to balance the amount of shuffled data and fair distribution of intermediate data on reduce nodes [23]. However, they implicitly assume the network connections between map tasks and reduce tasks are homogeneous and thus only consider the size of shuffled data. Their algorithm may result in significant partitioning skew in heterogeneous environments.

We propose a simple heuristic, which will be refined in the future, to determine where a reduce task should run. Given a reduce task that needs to fetch map output from a set of nodes where map tasks are located, we select the node which yields the minimum sum of data shuffling time and computation time. In other words, we pick the node that yields the earliest completion time. The execution of a reduce task cannot start until all assigned intermediate data are fetched. So the slowest data fetching determines the overall shuffling time. Because pairwise bandwidth information is maintained for all nodes and the amount of shuffled data is known after intermediate output is generated, we can calculate the data shuffling time for all candidate nodes where the reduce task under consideration can run potentially. Computation time can be estimated based on historical data if possible. If no historical data is available, sample run or user-provided hints are needed to provide approximately accurate estimation. In heterogeneous environments, we expect this heuristic to substantially outperform native Hadoop scheduling which randomly schedules reduce tasks.

	K_1	K_2	K_3	…	K_m
N_1	V_{11}	V_{12}	V_{13}	…	V_{1m}
N_2	V_{21}	…	…	…	…
N_3	V_{31}	…	…	…	…
…	…	…	…	…	…
N_n	V_{n1}	…	…	…	V_{nm}

Figure 3. Key ditribution of intermediate data on map side

3) Data Replication

According to a study [24], over 50% of data accesses take place 1 minute after its creation. During that short period, all replicas may not be fully created so that the data locality of subsequent MapReduce jobs will degrade. By default, HDFS maintains three replicas among which one replica is stored on the local rack and the other two are stored on a remote rack. This design reflects the philosophy "do not put all eggs in one basket" and thus has emphasis on data availability. Beyond these constraints, HDFS does not apply any optimization. One improvement we propose is that replicas should be quickly created by choosing the nodes with maximum interconnection throughput without violating the availability guarantee. In an environment with drastically heterogeneous network, full Hadoop replication may take long time. We propose that data availability can be violated temporarily by fully replicating data on nodes close to each other first and propagating it to a remote rack in background asynchronously. This can increase the data locality of successively submitted MapReduce jobs. Whenever data copies on a remote rack are created, the excessive replicas are removed to release storage space. Also the order of replica propagation can be tuned to speed up the replication process.

IV. Conclusion and Future Work

As some large-scale clusters exhibit inefficient resource utilization, it is possible to speed up task execution by using available resources more aggressively. We propose resource stealing to improve resource utilization by aggressively harnessing the portion of unutilized resources. Speculative execution in Hadoop was observed to be inefficient, which is caused by the excessive runs of useless speculative tasks. Our proposed Benefit Aware Speculative Execution manages speculative tasks in a benefit-aware manner and expected to improve the efficiency. Finally, various scheduling algorithms, which take into consideration the real-time network performance, are proposed to alleviate the performance degradation caused by network heterogeneity.

In the future, we will intensively evaluate the effectiveness of our approaches by running some non-trivial applications (e.g. sequence alignment algorithms Smith-Waterman-Gotoh and Needleman–Wunsch). In addition, we will further improve the proposed heterogeneity-aware scheduling heuristics of map and reduce tasks to make them more efficient and robust. As for data replication in HDFS, we will study how to calculate the best propagation path to minimize the replication time and thus improve the data locality of MapReduce jobs. Both storage and computation will be considered simultaneously to maximize the potential performance improvement.

ACKNOWLEDGMENT

This material is based upon work supported in part by the National Science Foundation under Grant No. 0910812.

REFERENCES

[1] S. Ghemawat, H. Gobioff, and S. T. Leung, "The google file system," SIGOPS Oper. Syst. Rev., vol. 37, no. 5, pp. 29-43, Oct. 2003. DOI: http://dx.doi.org/10.1145/1165389.945450

[2] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in Symposium on Operating System Design and Implementation (OSDI), 2004, pp. 137-150

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

[4] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," in Proc. Eur. Conf. Comput. Syst. (EuroSys), 2007, pp. 59–72

[5] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving MapReduce performance in heterogeneous environments," in Proceedings of the 8th USENIX conference on Operating systems design and implementation, ser. OSDI'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 29-42. [Online]. Available: http://portal.acm.org/citation.cfm?id=1855744

[6] X. Qiu, J. Ekanayake, S. Beason, T. Gunarathne, G. Fox, R. Barga, and D. Gannon, "Cloud technologies for bioinformatics applications," in Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers, ser. MTAGS '09. New York, NY, USA: ACM, 2009. [Online]. Available: http://dx.doi.org/10.1145/1646468.1646474

[7] C. T. Chu, S. K. Kim, Y. A. Lin, Y. Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun, "Map-reduce for machine learning on multicore," Advances in neural information processing systems, vol. 19, p. 281, 2007.

[8] 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 Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on, Apr. 2010, pp. 1-9. [Online]. Available: http://dx.doi.org/10.1109/IPDPSW.2010.5470880

[9] R. D. Blumofe and C. E. Leiserson, "Scheduling multithreaded computations by work stealing," in Proceedings of the 35th Annual Symposium on Foundations of Computer Science. Washington, DC, USA: IEEE Computer Society, 1994, pp. 356-368.

[10] S. N. Bhatt, F. R. K. Chung, F. T. Leighton, and A. L. Rosenberg, "On optimal strategies for Cycle-Stealing in networks of workstations," IEEE Trans. Comput., vol. 46, pp. 545-557, May 1997

[11] Z. Guo, M. Pierce, G. Fox, and M. Zhou, "Automatic task re-organization in MapReduce," in Proceedings of the 2011 IEEE International Conference on Cluster Computing, ser. CLUSTER '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 335-343. [Online]. Available: http://dx.doi.org/10.1109/CLUSTER.2011.44

[12] L. Gwennap. New algorithm improves branch prediction. Microprocessor Reports, March 27 1995.

[13] I. Ahmad and Y. K. Kwok, "A new approach to scheduling parallel programs using task duplication," in Proceedings of the 1994 International Conference on Parallel Processing - Volume 02, ser. ICPP '94. Washington, DC, USA: IEEE Computer Society, 1994, pp. 47-51.

[14] S. Seo, I. Jang, K. Woo, I. Kim, J.-S. Kim, and S. Maeng, "HPMR: Prefetching and pre-shuffling in shared MapReduce computation environment," in Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on, Sep. 2009, pp. 1-8. [Online]. Available: http://dx.doi.org/10.1109/CLUSTR.2009.5289171

[15] B. Palanisamy, A. Singh, L. Liu, and B. Jain, "Purlieus: locality-aware resource allocation for MapReduce in a cloud," in Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, ser. SC '11. New York, NY, USA: ACM, 2011. [Online]. Available: http://dx.doi.org/10.1145/2063384.2063462

[16] L. A. Barroso and U. Hölzle, "The case for Energy-Proportional computing," Computer, vol. 40, pp. 33-37, Dec. 2007

[17] Y. Luo, Z. Guo, Y. Sun, B. Plale, J. Qiu, and W. W. Li, "A hierarchical framework for cross-domain MapReduce execution," in Proceedings of the second international workshop on Emerging computational methods for the life sciences, ser. ECMLS '11. New York, NY, USA: ACM, 2011, pp. 15-22.

[18] M. Tsugawa and J. A. B. Fortes, "A virtual network (ViNe) architecture for grid computing," in Proceedings of the 20th international conference on Parallel and distributed processing, ser. IPDPS'06. Washington, DC, USA: IEEE Computer Society, 2006, p. 148.

[19] R. Wolski, N. T. Spring, and J. Hayes, "The network weather service: a distributed resource performance forecasting service for metacomputing," Future Gener. Comput. Syst., vol. 15, no. 5-6, pp. 757-768, Oct. 1999. [Online]. Available: http://dx.doi.org/10.1016/S0167-739X(99)00025-4

[20] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling," in Proc. of EuroSys. ACM, 2010, pp. 265-278.

[21] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," J. Parallel Distrib. Comput., vol. 59, pp. 107-131, Nov. 1999. [Online]. Available: http://portal.acm.org/citation.cfm?id=339727

[22] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," J. Parallel Distrib. Comput., vol. 61, no. 6, pp. 810-837, Jun. 2001. [Online]. Available: http://dx.doi.org/10.1006/jpdc.2000.1714

[23] S. Ibrahim, H. Jin, L. Lu, S. Wu, B. He, and L. Qi, "LEEN: Locality/Fairness-aware key partitioning for MapReduce in the cloud," in Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science, ser. CloudCom '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 17-24. [Online]. Available: http://dx.doi.org/10.1109/CloudCom.2010.25

[24] B. Fan, W. Tantisiriroj, L. Xiao, and G. Gibson, "DiskReduce: RAID for data-intensive scalable computing," in Proceedings of the 4th Annual Workshop on Petascale Data Storage, ser. PDSW '09. New York, NY, USA: ACM, 2009, pp. 6-10. [Online]. Available: http://dx.doi.org/10.1145/1713072.1713075