Daggupati Saidamma * et al.
(IJITR) INTERNATIONAL JOURNAL OF INNOVATIVE TECHNOLOGY AND RESEARCH
Volume No.5, Issue No.5, August - September 2017, 7339-7345.

Allocation of The Large Cluster Setups In Mapreduce

DAGGUPATI SAIDAMMA

Assistant Professor, Dept of CSE, Geetanjali College of Engineering and Technology

ABSTRACT: **Running multiple instances of the MapReduce framework concurrently in a multicluster system or datacenter enables data, failure, and version isolation, which is attractive for many organizations. It may also provide some form of performance isolation, but in order to achieve this in the face of time-varying workloads submitted to the MapReduce instances, a mechanism for dynamic resource (re-)allocations to those instances is required. In this paper, we present such a mechanism called Fawkes that attempts to balance the allocations to MapReduce instances so that they experience similar service levels. Fawkes proposes a new abstraction for deploying MapReduce instances on physical resources, the MR-cluster, which represents a set of resources that can grow and shrink, and that has a core on which MapReduce is installed with the usual data locality assumptions but that relaxes those assumptions for nodes outside the core. Fawkes dynamically grows and shrinks the active MRcluster based on a family of weighting policies with weights derived from monitoring their operation. Implementing MapReduce in cloud requires creation of clusters, where the Map and Reduce operations can be performed. Optimizing the overall resource utilization without compromising with the efficiency of availing services is the need for the hour. Selecting right set of nodes to form cluster plays a major role in improving the performance of the cloud. As a huge amount of data transfer takes place during the data analysis phase, network latency becomes the defining factor in improving the QoS of the cloud. In this paper we propose a novel Cluster Configuration algorithm that selects optimal nodes in a dynamic cloud environment to configure a cluster for running MapReduce jobs. The algorithm is cost optimized, adheres to global resource utilization and provides high performance to the clients. The proposed Algorithm gives a performance benefit of 35% on all reconfiguration based cases and 45 % performance benefit on best cases.**

Index Terms— **Mapreduce; Cloud Computing; Hadoop; Distributed Computing;**

I. INTRODUCTION

In last decade, scientific research trend have become increasingly reliant on processing huge volume of data. The data inflow has come from various fields such as Social Media, Weather Service Centre and Organization such as Newyork Stock Exchange. People send large volume of unstructured data in the form of messages and photographs. People have started creating tools to use and analyze these data. The overall impact of all these can be seen in near future. Since the size of the data is growing at a much higher rate than the rate of accessing the data [18], it became very important to create new system which can handle huge volume of data without deteriorating the performance. New concepts like Cloud computing, MapReduce and its open source implementation Hadoop became widely accepted for doing large scale data analysis. While cloud computing offers raw computing power in the form of storage and other services, there is a requirement of distributed framework to harness the power of cloud easily and efficiently. Google in 2004 introduced a model known as MapReduce [19], which was capable of handling execution of large distributed jobs in cloud infrastructure. It has been found that operating cost of data centers have doubled in last 5 years and 75% of investment has been on infrastructure and energy consumption [5]. Gartner's survey [8] shows that enterprises invest 39% of their IT budget in Cloud. Improving the global resource utilization can reduce the overall infrastructure cost. Selecting the right set of nodes to form a cluster is one of the prime factors resulting in improved global resource utilization. Network Latency is one of the major factors of energy consumption in a cloud [17]. Hence we propose a Dynamic Cluster Configuration algorithm which reduces the network latency and can improve the utilization of the cloud resources without compromising with the efficiency of the cloud.

To reduce network traffic for MapReduce workloads, we argue for improved data locality for both Map and Reduce phases of the job. The goal is to reduce the network distance between storage and compute nodes for both map and reduce processing – for map phase, the VM executing the map task should be close to the node that stores the input data (preferably local to that node) and for reduce phase, the VMs executing reduce tasks should be close to the map-task VMs which generate the intermediate data used as reduce input. Improved data locality in this manner is beneficial in two ways – (1) it reduces job execution times as network transfer times are big components of total execution time and (2) it reduces cumulative data center network traffic. While map locality is well understood and implemented in MapReduce systems, reduce locality has surprisingly received little attention in spite of its significant potential impact. As an example, Figure 1 shows the impact

of improved reduce locality for a Sort workload. It shows the Hadoop task execution timelines for a 10 GB dataset in a 2-rack 20-node physical cluster1 , where 20 Hadoop VMs were placed without and with reduce locality (top and bottom figures respectively). As seen from the graph, reduce locality resulted in a significantly shorter shuffle phase helping reduce total job runtime by 4x. In this paper, we present Purlieus – an intelligent MapReduce cloud resource allocation system. Purlieus improves data locality during both map and reduce phases of the MapReduce job by carefully coupling data and computation (VM) placement in the cloud. Purlieus categorizes MapReduce jobs based on how much data they access during the map and reduce phases and analyzes the network flows between sets of machines that store the input/intermediate data and those that process the data. It places data on those machines that can either be used to process the data themself or are close to the machines that can do the processing. This is in contrast to conventional MapReduce systems which place data independent of map and reduce computational placement – data is placed on any node in the cluster which has sufficient storage capacity [3, 23] and only map tasks are attempted to be scheduled local to the node storing the data block. Additionally, Purlieus is different from conventional MapReduce clouds (e.g., Amazon Elastic MapReduce [14]) that use a separate compute cloud for performing MapReduce computation and a separate storage cloud for storing the data persistently. Such an architecture delays job execution and duplicates data in the cloud. In contrast, Purlieus stores the data in a dedicated MapReduce cloud and jobs execute on the same machines that store the data without waiting to load data from a remote storage cloud. To the best of our knowledge, Purlieus is the first effort that attempts to improve data locality for MapReduce in a cloud. Secondly, Purlieus tackles the locality problem in a fundamental manner by coupling data placement with VM placement to provide both map and reduce locality. This leads to significant savings and can reduce job execution times by close to 50% while reducing up to 70% of cross-rack network traffic in some scenarios.

II. BACKGROUND

Scheduling in Hadoop In this section, we describe the mechanism used by Hadoop to distribute work across a cluster. We identify assumptions made by the scheduler that hurt its performance. These motivate our LATE scheduler, which can outperform Hadoop's by a factor of 2. Hadoop's implementation of MapReduce closely resembles Google's [1]. There is a single master managing a number of slaves. The input file, which resides on a distributed filesystem throughout the cluster, is split into even-sized chunks replicated for fault-tolerance. Hadoop divides each MapReduce job into a set of tasks. Each chunk of input is first processed by a map task, which outputs a list of key-value pairs generated by a userdefined map function. Map outputs are split into buckets based on key. When all maps have finished, reduce tasks apply a reduce function to the list of map outputs with each key. Figure 1 illustrates a MapReduce computation. Hadoop runs several maps and reduces concurrently on each slave – two of each by default – to overlap computation and I/O. Each slave tells the master when it has empty task slots. The scheduler then assigns it tasks. The goal of speculative execution is to minimize a job's response time. Response time is most important for short jobs where a user wants an answer quickly, such as queries on log data for debugging, monitoring and business intelligence. Short jobs are a major use case for MapReduce. For example, the average MapReduce job at Google in September 2007 took 395 seconds [1]. Systems designed for SQL-like queries on top of MapReduce, such as Sawzall [9] and Pig [10], underline the importance of MapReduce for ad-hoc queries. Response time is also clearly important in a pay-by-the-hour environment like EC2. Speculative execution is less useful in long jobs, because only the last wave of tasks is affected, and it may be inappropriate for batch jobs if throughput is the only metric of interest, because speculative tasks imply wasted work. However, even in pure throughput systems, speculation may be beneficial to prevent the prolonged life of many concurrent jobs all suffering from straggler tasks. Such nearly complete jobs occupy resources on the master and disk space for map outputs on the slaves until they terminate. Nonetheless, in our work, we focus on improving response time for short jobs.

Based on these observations, we investigate intelligent data placement as a potential avenue to reduce remote accesses. We focus our investigation on the "map" phase of MapReduce jobs as initial data placement is immaterial thereafter. Hadoop's scheduler is designed to assign map tasks to nodes such that they access data locally whenever possible. When a computation resource is assigned to a job, the scheduler scans the list of incomplete map tasks for that job to find any tasks that can access locally available data. Only if no such tasks are available will it schedule a task that must perform remote accesses. Hence, jobs with dedicated access to the entire cluster rarely incur remote accesses (remote accesses only arise at the end of the map phase, when few map tasks remain, or under substantial load imbalance, for example, due to server heterogeneity [4]). However, restrictions on task assignment, because of long-running tasks, prioritization among competing jobs,

dedicated allocations, or other factors, can rapidly increase the number of remote accesses. We contrast Hadoop's default random data placement policy against an extreme alternative, partitioned data placement, wherein a cluster is divided into partitions, each of which contains one replica of each data block. (Note that, since the number of replicas is unchanged and placement remains random within each partition, availability is, to first-order, unchanged). By segregating replicas, due simply to combinatorial effects, we increase the probability that a large fraction of distinct data blocks is available even within relatively small, randomly selected allocations of the cluster.

We further consider the utility of adding additional replicas for frequently accessed blocks, to increase the probability that these blocks will be available locally in a busy cluster. Our evaluation, through a combination of simulation of the Hadoop scheduling algorithm and validation on a small-scale test cluster, leads to mixed conclusions:

• When scheduling is unconstrained and task lengths are well-chosen to balance load and avoid long-running tasks, Hadoop's scheduler is highly effective in avoiding remote accesses regardless of data placement, as the job can migrate across nodes over time to process data blocks locally. Under an "Unconstrained" allocation scenario, Hadoop can achieve 98% local accesses.

• However, when task allocation is constrained to a subset of the cluster (e.g., because of long-running tasks, reserved nodes, restrictions arising from job priorities, power management [16], or other node allocation constraints), partitioned data placement substantially reduces remote data accesses. For example, under a "Restricted" allocation scenario where a job may execute on only one-third of nodes (selected at random), partitioned data placement reduces remote accesses by 86% over random data placement.

• We demonstrate that selective replication of frequently accessed blocks can further reduce remote accesses in restricted allocation scenarios

III. RELATED WORK

Data replication is widely used in distributed systems to improve performance when a system needs to scale in numbers and/or geographical area [23]. Replication can increase data availability, and helps achieve load balancing in the presence of scaling. For geographically dispersed systems, replication can reduce communication latencies. Hadoop leverages replication to provide both availability and scalability. Further, Hadoop places two replicas of a data block on the same rack to save inter-rack bandwidth. Caching is a special form of replication where a copy of the data under consideration is placed close to the client that is accessing the data. Caching has been used effectively in distributed file systems such as the Andrew File System (AFS) and Coda to minimize network traffic [12], [21].

Gwertzman and Seltzer have proposed a technique of server-initiated caching called push caching [11]. Under this technique, a server places temporary replicas of data closer to geographical regions from which large fractions of requests are arriving. Since replication and caching imply multiple copies of a data resource, modification of one copy creates consistency issues. Much research in the distributed systems field has been devoted to efficient consistency maintenance [19], [23]. However, since Hadoop follows a write-once, read-many model for data (i.e., data files are immutable), maintaining consistency is not a concern. In systems with distributed data replicas, achieving locality while maintaining fairness is a challenge. Isard and co-authors propose Quincy, a framework for scheduling concurrent distributed jobs with fine-grain resource sharing [13]. Quincy defines fairness in terms of disk-locality and can evict tasks to ensure fair distribution of disk-locality across jobs. Overall, the system improves both fairness and locality, achieving a 3.9x reduction in the amount of data transferred and a throughput increase of up to 40%.

Zaharia et al. create a fair-scheduler that maintains task locality and achieves almost 99% local accesses via delay scheduling [25]. Under delay scheduling, when a job that should be scheduled next under fair-scheduling cannot launch a data-local task, it stalls a small amount of time while allowing tasks from other jobs to be scheduled. However, delay scheduling performs poorly in the presence of long tasks (nodes do not free up frequently enough for jobs to achieve locality) and hotspots (certain nodes are of interest to many jobs; for example, such nodes might contain a data block that many jobs require). The authors suggest long-taskbalancing and hotspot replication as potential solutions, but do not implement either. In contrast to the authors' approach, we focus on how intelligent data placement can be used to maximize MapReduce efficiency in scenarios where node allocations are restricted.

Eltabakh and co-authors present CoHadoop [9], a lightweight extension of Hadoop that allows applications to control where data are stored. Applications give hints to CoHadoop that certain files are related and may be processed jointly; CoHadoop then tries to co-locate these files for improved efficiency. Ferguson and Fonseca [10] highlight the non-uniformity in data placement within Hadoop clusters, which can lead to performance degradation. They propose placing

data on nodes in a round-robin fashion instead of Hadoop's default data placement, and demonstrate an 11.5% speedup for the sort benchmark. Ahmad et al. [4] observe that MapReduce's built-in load balancing results in excessive and bursty network traffic, and that heterogeneity amplifies load imbalances. In response, the authors develop Tarazu, a set of optimizations to improve MapReduce performance on heterogeneous clusters.

Xie et al. [24] study the effect of data placement in clusters of heterogeneous machines, and suggest placing more data on faster nodes to improve the percentage of local accesses. Zaharia et al. [26] also investigate MapReduce performance in heterogenous environments. The authors design a scheduling algorithm called Longest Approximate Time to End (LATE), that is robust to heterogeneity and can improve Hadoop response times by a factor of two. Ananthanarayanan et al. [5] observe that MapReduce frameworks use filesystems that replicate data uniformly to improve data availability and resilience. However, job logs from large production clusters show a wide disparity in data popularity.

The authors observe that machines and racks storing popular content become bottlenecks, thereby increasing the completion times of jobs accessing these data even when there are machines with spare cycles in the cluster. To address this problem, the authors propose a system called Scarlett. Scarlett accurately predicts file popularity using learned trends, and then selectively replicates blocks based on their popularity. In trace driven simulations and experiments on Hadoop and Dryad clusters, Scarlett alleviates hotspots and speeds up jobs by up to 20.2%. We explore the utility of selective replication in combination with partitioned data placement in subsequent sections.

IV. PURLIEUS: PLACEMENT TECHNIQUES

Next, we describe Purlieus's data and VM placement techniques for various classes of MapReduce jobs. The goal of these placements is to minimize the total Cost by reducing the dist function for map (when input data, Qi is large) and/or reduce (when intermediate data, mout is large).

4.1 Map-input heavy jobs

Map-input heavy jobs read large amounts of input data for map but generate only small map-outputs that is input to the reducers. For placement, mappers of these jobs should be placed close to input data blocks so that they can read data locally, while reducers can be scheduled farther since amount of map-output data is small.

4.1.1 Placing Map-input heavy data

As map-input heavy jobs do not require reducers to be executed close to each other, the VMs of the MapReduce cluster can be placed anywhere in the data center. Thus, physical machines to place the data are chosen only based on the storage utilization and the expected load, Ek on the machines. As discussed in the cost model, E k denotes the expected load on machine, Mk. E k = X i Wk i × CRes(Di) To store map-input heavy data chunks, Purlieus chooses machines that have the least expected load. This ensures that when MapReduce VMs are placed, there is likely to be capacity available on machines storing the input data. 4.1.2 VM placement for Map-input heavy jobs The VM placement algorithm attempts to place VMs on the physical machines that contain the input data chunks for the map phase. This results in lower MCost – the dominant component for map-input heavy jobs. Since data placement had placed blocks on machines that have lower expected computational load, it is less likely, though possible that at the time of job execution, some machine containing the data chunks does not have the available capacity. For such a case, the VM may be placed close to the node that stores the actual data chunk. Specifically, the VM placement algorithm iteratively searches for a physical machine having enough resources in increasing order of network distance from the physical machine storing the input data chunk. Among the physical machines at a given network distance, the one having the least load is chosen.

4.2 Map-and-Reduce-input heavy jobs

Map-and-reduce-input heavy jobs process large amounts of input data and also generate large intermediate data. Optimizing cost for such jobs requires reducing the dist function during both their map and reduce phases.

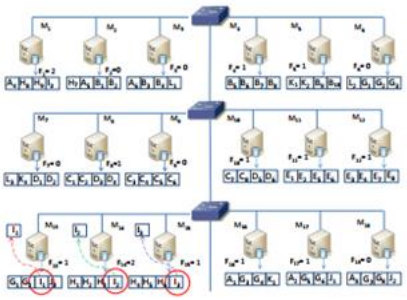
4.2.1 Placing Map-and-Reduce-input heavy data

To achieve high map-locality, data should be placed on physical machines that can host VMs locally. Additionally, this data placement should support reduce-locality – for which the VMs should be hosted on machines close to each other (preferably within the rack) so that reduce traffic does not significantly load the data center network. Ideally, a subgraph structure that is densely connected, similar to a clique, where every node is connected to every other node in 1-hop would be a good candidate for placing the VMs. However, it may not always be possible to find cliques of a given size as the physical network may not have a clique or even if it does, some of the machines may not have enough resources to hold the data or their expected computational load may be high to not allow VM placement later. An alternate approach

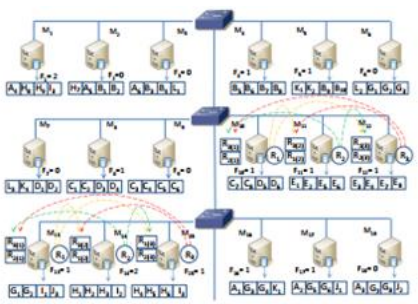
would be to find subgraph structures similar to cliques. A number of clique relaxations have been proposed, one of which is k-club [27]. A k-club of a graph G is defined as a maximal subgraph of G of diameter k. While finding k-club is NP-Complete for a general graph, data center networks are typically hierarchical (e.g. fat-tree topologies) and this allows finding a k-club in polynomial time. In a data center tree topology, the leaf nodes represent the physical machines and the non-leaf nodes represent the network switches. To find a k-club containing n leaf nodes, the algorithm simply finds the sub-tree of height k 2 containing n or more leaf nodes. For map-and-reduce-input heavy jobs, data blocks get placed in a set of closely connected physical machines that form a k-club of least possible k (least possible height of the subtree) given the available storage resources in them. If several subtrees exists with the same height, then the one having the maximum available resource is chosen. As an illustration, in Figure 4(a), the input data blocks, I1, I2, and I3 are stored in a closely connected set of nodes M13, M14 and M15 that form a k-club of least possible k in the cluster.

4.2.2 VM placement for Map and Reduce-input heavy jobs

As data placement had done an optimized placement by placing data blocks in a set of closely connected nodes, VM placement algorithm only needs to ensure that VMs get placed on either the physical machines storing the input data or the close-by ones. This reduces the distance on the network that the reduce traffic needs to go over, speeding up job execution while simultaneously reducing cumulative data center network traffic. In the example shown



(a) Map-phase



(b) Reduce-phase

Figure 4: Data and VM placement. Bottom squares show data blocks placed on each machine. Squares next to a machine (e.g. I1 near M13 for map-phase in figure 4(a)) indicates reading of the block for map processing. F measure denotes available computational capacity – for simplicity, number of VMs that can be placed on that machine. In reduce phase (figure 4(b)) , circled Ri indicates map outputs and square Ri(j) indicates reading of intermediate data for reducer j from map task output, in Figure 4, VMs for job on dataset I get placed on the physical machines storing input data. As a result, map tasks use local reads (Figure 4(a)) and reduce tasks also read within the same rack, thereby maximizing reduce locality (Figure 4(b)). In case node M15 did not have available resources to host the VM, then the next candidates to host the VM would be M16, M17 and M18, all of which can access the input data block I3 by traversing one network switch and are close to the other reducers executing in M13 and M14. If any of M16, M17 and M18 did not have available resources to host a new VM, then the algorithm would iteratively proceed to the next rack (M7, M8, M9, M10, M11 and M12) and look for a physical machine to host the VM. Thus the algorithm tries to maximize locality even if the physical machines containing input data blocks are unavailable to host the VMs.

V. CONCLUSION

This paper presents Purlieus, a resource allocation system for MapReduce in a cloud. We present a system architecture for the MapReduce cloud service and describe how existing data and virtual machine placement techniques lead to longer job execution times and large amounts of network traffic in the data center. We identify data locality as the key principle which if exploited can alleviate these problems and develop a unique coupled data and VM placement technique that achieves high data locality. Uniquely, Purlieus's proposed placement techniques optimize for data locality during both map and reduce phases of the job by considering VM placement, MapReduce job characteristics and load on the physical cloud infrastructure at the time of data placement. Our detailed evaluation shows significant performance gains with some scenarios showing close to 50% reduction in execution time and upto 70% reduction in the cross-rack network traffic. We plan to extend our work in two directions. First, for placement techniques we would like to capture relationships between datasets, e.g. if two datasets are accessed together (MapReduce job doing a join of two datasets), their data placement can be more intelligent while placing their blocks in relation to each other. Second, we plan to develop online techniques to handle dynamic scenarios like

changing job characteristics on a dataset. While core principles developed in this work will continue to apply, such scenarios may use other virtualization technologies like live data and VM migration.

VI. REFERENCES

[1] B. Igou "User Survey Analysis: Cloud-Computing Budgets Are Growing and Shifting; Traditional IT Services Providers Must Prepare or Perish". Gartner Report, 2010

[2] http://en.wikipedia.org/wiki/Loop_device

[3] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In OSDI, 2004.

[4] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha and E. Harris. Reining in the Outliers inMap-Reduce Clusters using Mantri. In OSDI, 2010.

[5] http://en.wikipedia.org/wiki/Big-data

[6] S. Babu. Towards Automatic Optimization of MapReduce Programs. In SOCC, 2010.

[7] http://en.wikipedia.org/wiki/Clickstream

[8] K. Kambatla, A. Pathak and H. Pucha. Towards Optimizing Hadoop Provisioning in the Cloud. In HotCloud, 2009.

[9] Cloudera. http://www.cloudera.com/blog/2010/08/hadoopfor-fraud-detection-and-prevention/

[10] K. Morton, A. Friesen, M. Balazinska, D. Grossman. Estimating the Progress of MapReduce Pipelines. In ICDE, 2010.

[11] S. Ghemawat, H. Gobioff and S. Leung, "The Google File System," ACM SIGOPS Operating Systems Review, vol. 37, no. 5, pp. 29–43, 2003.

[12] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters" ACM Commun., vol. 51, pp. 107–113, 2008.

[13] J. Li, P. Roy, S. U. Khan, L. Wang and Y. Bai, "Data mining using clouds: an experimental implementation of apriori over mapreduce," http://sameekhan.org/pub/L−K−2012−SCALCOM.pdf

[14] X. Lin, "MR-Apriori: association rules algorithm based on mapreduce," IEEE, 2014.

[15] X. Y. Yang, Z. Liu and Y. Fu, "MapReduce as a programming model for association rules algorithm on hadoop," in Proceedings 3rd International Conference on Information Sciences and Interaction Sciences (ICIS), 2010, vol. 99, no. 102, pp. 23–25.

[16] N. Li, L. Zeng, Q. He and Z. Shi, "Parallel implementation of apriori algorithm based on mapreduce," in Proceedings 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing, IEEE, 2012, pp. 236–241.

[17] S. Oruganti, Q. Ding and N. Tabrizi, "Exploring Hadoop as a platform for distributed association rule mining," in FUTURE COMPUTING 2013 the Fifth International Conference on Future Computational Technologies and Applications, pp. 62–67.

[18] M-Y. Lin, P-Y. Lee and S-C. Hsueh, "Apriori-based frequent itemset mining algorithms on mapreduce," in Proceedings 6th International Conference on Ubiquitous Information Management and Communication (ICUIMC '12), ACM, New York, 2012, Article 76.

[19] F. Kovacs and J. Illes, "Frequent itemset mining on Hadoop," in Proceedings IEEE 9th International Conference on Computational Cybernetics (ICCC), Hungry, 2013, pp. 241–245.

[20] L. Li and M. Zhang, "The strategy of mining association rule based on cloud computing," in Proceedings IEEE International Conference on Business Computing and Global Informatization (BCGIN), 2011, pp. 29–31.

